



易  
学  
易  
用

# 中文编程

## 从入门到精通

大连易翔软件开发有限公司 编著

- 中文编程语言 易学易用实用
- 全中文 全可视 跨平台编程
- 模块化开发 面向对象编程
- 内置输入法 即时信息帮助
- 可扩展支持库 支持所有数据库
- 即看即学 轻松学会编程
- 图文并茂 代码明晰
- 行业程序源码解读 游戏开发设计分析



海洋出版社



责任编辑：张鹤凌  
封面设计：姜 岩

WISBOOK®  
智慧图书

# 5天学会编程

本书全面讲解全中文、全可视、全编译、跨平台的编程语言——“易语言”。“易语言”实现了真正的中文编程，彻底摆脱英文的语言模式和西方的思维模式，符合中国人的习惯，只要会汉字输入或拼音输入，无需任何编程基础即可轻松学会编程，“易语言”功能强大，资源丰富，将是广大编程爱好者的最理想的选择。

本书详细介绍了易语言的使用方法，内容详实、通俗易懂、结构清晰、循序渐进、图文并茂，配有大量示例。

全书以五大部分，十七章组成，百个学习例程，千张图片演示，万行代码教学，完整行业程序源码解读，游戏开发设计与分析。每一章节学习完成，都可以制作一个软件，伴随学习的加深，逐渐完善程序功能，且配章节小结、习题，来加以检查学习的掌握程度。在读完本书后，读者将具备独立的分析、编程能力，且掌握编程中常用技巧，程序纠错调整与修改。

数万学习源代码：<http://www.wodesoft.com>

易语言官方论坛：<http://bbs.eyuyan.com>

ISBN 978-7-5027-7931-3



9 787502 779313 >

定价：86.00元（含1CD）

更多书籍：[www.i-ebook.cn](http://www.i-ebook.cn)





易  
語  
言

# 中文编程 从入门到精通

大连易翔软件开发有限公司 编著

中文编程语言 易学易用实用  
全中文 全可视 跨平台编程  
模块化开发 面向对象编程  
内置输入法 即时信息帮助  
可扩展支持库 支持所有数据库

即看即学 轻松学会编程  
图文并茂 代码明晰  
行业程序源码解读 游戏开发设计分析

海洋出版社

2011年·北京

更多书籍：[www.i-ebook.cn](http://www.i-ebook.cn)



## 内 容 简 介

本书介绍了全中文、全可视、全编译、跨平台的编程语言——“易语言”。该程序实现了真正的中文编程，彻底摆脱英文的语言模式，符合中国人的习惯。“易语言”功能强大，资源丰富，是广大编程爱好者的最理想的选择。书中全面、详细介绍了易语言的使用方法，内容详实、通俗易懂、结构清晰、循序渐进、图文并茂，配有大量示例。

全书分五大部分，共十七章，包括上百个学习例程，数千张图片演示，近万行代码教学，完整行业程序源码解读，游戏开发设计与分析。每一章节学习完成，都可以制作一个软件，伴随学习的加深，逐渐完善程序功能。每章后还配有章节小结、习题，以检查学习的掌握程度。

读者将具备独立的分析、编程能力，且掌握编程中常用技巧，程序纠错调整与修改。

### 图书在版编目(CIP)数据

中文编程·从入门到精通/大连易翔软件开发有限公司编著. —北京: 海洋出版社, 2011.1  
ISBN 978-7-5027-7931-3

I. ①中… II. ①大… III. ①程序设计 IV. ①TP311

中国版本图书馆 CIP 数据核字 (2010) 第 232276 号

总 策 划: 邹华跃

责任编辑: 张鹤凌

责任校对: 肖新民

责任印制: 刘志恒

排 版: 海洋计算机图书输出中心 晓阳

出版发行: 海洋出版社

地 址: 北京市海淀区大慧寺路 8 号 (705 房间)  
100081

经 销: 新华书店

技术支持: (010) 62100057

发 行 部: (010) 62174379 (传真) (010) 62132549

(010) 62100075 (邮购) (010) 62173651

网 址: [www.oceanpress.com.cn](http://www.oceanpress.com.cn)

承 印: 北京盛兰兄弟印刷装订有限公司

版 次: 2011 年 1 月第 1 版

2011 年 1 月第 1 次印刷

开 本: 787mm×1092mm 1/16

印 张: 45.25 (彩插 2 页)

字 数: 1123 千字

印 数: 1~3000 册

定 价: 86.00 元 (含 1CD)

本书如有印、装质量问题可与发行部调换



# 序

受大连易翔软件开发有限公司全体编者诚挚邀请为本书作序，我作为易语言的创始人，很开心，也很欣慰。开心的是用户又多了一本全面学习易语言的资料；欣慰的是更多的人在为易语言的推广而努力奋斗。

研发易语言至今有十年的时间，组建易语言公司也有六年之久，在此期间伴随研发团队的壮大，易语言在功能上不断完善；编辑环境更加规范化；编译方式也有了很大的改进。2004年11月，易语言通过了国家的软件技术鉴定，成为了国家教育部中小学计算机初级教育的首选教育软件；且成功地参加了大连国际软件交易会，使易语言的知名度不断地扩大，更为易语言成为中国最好的中文编程语言及编程环境奠定了良好的基石。

易语言公司在2005年曾出版了一本《易语言编程系统》，但易语言的快速发展，版本的不断更新、升级，相关辅助教材也需要继续完善升级，虽然期间也曾持续编写了不少教材，但为了能降低用户的学习成本，基本都在易语言论坛发布了电子版。

如今大连易翔软件开发有限公司作为易语言核心代理孵化公司率先为大家编写了一本全新版本的易语言书籍，是值得大家关注和支持的。根据我与编者的交谈，了解到本书是围绕易语言核心支持库，全面、完整地讲解了易语言的使用与提高，内容详实、结构清晰、循序渐进，并注重各章节与实例之间的呼应、对照，是一本非常不错的易语言编程入门、提高教材。尤其值得一提的是本书的最后部分，对这一行业应用软件完整剖析，值得大家研究、学习，同时本书也不失娱乐中轻松学会易语言的目的，利用核心支持库编写一款小游戏，并加以详细解读。

易语言十年来的努力和发展得到了众多人的鼎力支持，期间经历了不少风雨，前方虽然还有荆棘，但是易语言仍在继续前进，没有什么能够阻挡它的脚步；借此机会由衷地感谢长期以来支持和关注易语言发展的各届朋友，正因为有大家不懈的支持与关注易语言才能勇往直前；也感谢易翔软件开发有限公司为易语言的发展做出的努力，同时也希望更多有梦想、有激情的团队加入到易语言前进步伐中来。

我们的梦想是一致的：就是让神州大地上从事各种各样职业的人们，能够轻松拿起易语言这只笔，在电脑屏幕上尽情挥洒，让电脑传承下他们丰富的业务经验、让他们驾驭这匹快马，风驰电掣、美梦成真！

愿中华民族永远长青！

吴 涛  
2010年10月



# 前 言

在本书中，读者将学习如何使用易语言这一极为灵活和完备的编程工具，去建立各种各样的应用程序。通过对每一章的了解、学习，将逐渐掌握对易语言的操作与应用，伴随各部分功能例程的实践配合每章小结与习题，来检验学习的进度与知识点的掌握。为了更好地了解、学习本书，首先介绍一下本书各部分的知识要点：

本书整体分为五大部分

## 第一部分：易语言的基础知识

易语言的特点；怎样安装易语言；易语言窗口界面的设计使用；菜单栏的功能；代码编辑环境的使用；易语言编程基础知识（常量、变量、表达式、子程序的使用）以及第一个易程序的编写等。

## 第二部分：易语言的命令与组件

本部分将介绍易语言核心支持库中的命令与组件的使用。包括：各命令的调用、解释，各组件的属性、方法、事件等。

## 第三部分：易语言中的数据库应用

本部分将介绍易语言的数据库的应用。您将学习以下内容：易语言自带数据库的应用；易语言通过ODBC和ADO方式调用外部数据库的应用。

## 第四部分：易语言的高级应用

本部分将介绍如何编写易语言模块、DLL、向导程序；如何调用模块、API、DLL、向导程序；如何调用外部OCX组件和类型库；了解对象和面向对象；以及程序的调试和编译发布。

## 第五部分：易语言的综合实例

本部分将介绍学习到完整的易语言程序，在了解程序的设计编写思路后，理解程序代码的编写，从而学会如何编写一个较大的应用程序。

其次介绍一下五天学会易语言的具体流程，从而使初学者能快速入门。

本书第一章是易语言的基础知识，因此建议大家完整阅读与学习，阅读中可以对应软件具体加以参照，本章结尾将学会编写“我的播放器（一）”。

在第二章中主要对易语言的编程基础知识加以简单的了解与熟悉，这里需要对“2.1.1 数据类型”、“2.1.2 变量”、“2.3 子程序的编写与调用”加以学习。

在第三章中主要围绕“我的播放器（三）”相关内容加以简单的了解与熟悉，这里需要对“3.1 了解易语言命令”、“3.2 流程控制命令”、“3.16 媒体播放命令”加以了解即可。



在第四章中主要围绕“我的播放器（四）”相关内容加以简单的了解与熟悉，这里需要对“4.1 窗口”、“4.3 按钮类组件”加以了解即可。

至此通过以上的学习可以说对易语言已经有了基本的了解与熟悉，从而对再次深入学习做好铺垫，本书第一章、第二章是易语言的最基础的知识，大家要多对比实践加深了解，这样才能在后期的学习中运用自如。

当结束了本书的全部内容，读者则具备了易语言的核心知识与应用，同时也具备了编程知识和技巧，并足以编写大多数WINDOWS应用程序。在具体应用与学习其他类数据库时可以首先参考阅读其说明文档，然后参考相关例程加以实践即可快速掌握与运用，同时在“易语言官方论坛（[bbs.eyuyan.com](http://bbs.eyuyan.com)）”与“易语言资源网（[www.wodesoft.com](http://www.wodesoft.com)）”中有很多源码，大家在学习中可以加以借鉴与参考，从而能更快、更好、更灵活地掌握和使用易语言这一中国人自己的编程语言。

编 者

2010. 10. 23



# 目 录

## 第一部分 易语言的基础知识

第一章 概述.....	2	第二章 易语言编程基础.....	37
1.1 易语言概述 .....	3	2.1 变量与常量 .....	38
1.1.1 易语言简介 .....	3	2.1.1 数据类型.....	38
1.1.2 易语言的发展史.....	4	2.1.2 变量.....	44
1.1.3 易语言的特点.....	4	2.1.3 常量.....	54
1.2 易语言的安装 .....	9	2.1.4 资源表.....	62
1.3 易语言的工作界面 .....	10	2.2 运算符和表达式 .....	65
1.3.1 易语言的界面.....	10	2.2.1 运算符.....	66
1.3.2 易语言的菜单栏.....	13	2.2.2 表达式.....	67
1.4 代码编辑环境 .....	25	2.2.3 赋值运算符和赋值表达式.....	68
1.4.1 代码输入提示.....	25	2.3 子程序的编写与调用 .....	68
1.4.2 前层提示信息.....	26	2.3.1 了解子程序.....	68
1.4.3 代码输入方式.....	26	2.3.2 事件子程序.....	69
1.4.4 参数分步输入.....	27	2.3.3 用户自定义子程序.....	70
1.4.5 输入注释与代码屏蔽.....	28	2.3.4 子程序的返回值.....	71
1.4.6 易语言语句分类.....	29	2.3.5 子程序的调用.....	72
1.4.7 易语言中的关键字.....	29	2.3.6 子程序的参数.....	73
1.4.8 书签.....	29	2.3.7 子程序的递归调用.....	79
1.4.9 即时帮助和帮助文档.....	30	2.4 我的播放器（二） .....	82
1.5 第一个易语言程序 .....	32	2.5 小结 .....	83
1.6 我的播放器（一） .....	34	2.6 习题 .....	84
1.7 小结 .....	36		

## 第二部分 易语言的命令与组件

第三章 易语言的命令 .....	86	3.1.4 易语言命令的返回值.....	88
3.1 了解易语言命令 .....	87	3.1.5 易语言命令嵌套调用.....	90
3.1.1 易语言命令概述.....	87	3.1.6 数组参数与数组返回值.....	90
3.1.2 易语言命令的格式.....	87	3.2 流程控制命令 .....	91
3.1.3 易语言命令的参数.....	87	3.2.1 了解流程控制类命令.....	91





3.2.2 分支类流程控制命令 .....	92	3.17.3 易语言中的网络通信命令 .....	196
3.2.3 循环类流程控制命令 .....	94	3.18 其他命令 .....	197
3.2.4 跳转类流程控制命令 .....	98	3.19 我的播放器（三） .....	213
3.3 算术运算命令 .....	100	3.20 小结 .....	215
3.3.1 基本算术运算命令及其运算符 ....	100	3.21 习题 .....	215
3.3.2 扩展算术运算命令 .....	102	<b>第四章 易语言组件</b> .....	216
3.4 逻辑比较 .....	105	4.1 窗口 .....	217
3.4.1 逻辑比较命令 .....	106	4.1.1 窗口的定义 .....	217
3.4.2 多条件逻辑比较时的运算顺序 ....	108	4.1.2 共有属性 .....	217
3.5 位运算命令 .....	109	4.1.3 独有属性 .....	222
3.5.1 了解位运算 .....	109	4.1.4 共有事件 .....	228
3.5.2 常用进制 .....	109	4.1.5 独有事件 .....	237
3.5.3 位运算命令 .....	111	4.1.6 共有命令 .....	241
3.6 数组操作命令 .....	115	4.2 菜单 .....	254
3.6.1 了解数组 .....	115	4.2.1 创建菜单 .....	254
3.6.2 数组操作命令 .....	115	4.2.2 菜单的热键及属性 .....	255
3.7 环境存取命令 .....	119	4.3 按钮类组件 .....	256
3.8 拼音处理命令 .....	121	4.3.1 按钮 .....	256
3.9 文本操作命令 .....	124	4.3.2 图形按钮 .....	258
3.9.1 文字编码和存储方式 .....	124	4.4 列表类组件 .....	261
3.9.2 ASCII码 .....	124	4.4.1 组合框 .....	261
3.9.3 区别键代码和文字编码 .....	128	4.4.2 列表框 .....	273
3.9.4 文本操作命令 .....	129	4.4.3 选择列表框 .....	284
3.10 时间操作命令 .....	136	4.5 系统类组件 .....	294
3.11 数值转换命令 .....	143	4.5.1 通用对话框 .....	294
3.12 字节集操作命令 .....	147	4.5.2 文件框 .....	302
3.13 磁盘操作命令 .....	153	4.5.3 目录框 .....	307
3.13.1 相关知识 .....	153	4.5.4 驱动器框 .....	308
3.13.2 易语言中的磁盘操作命令 .....	153	4.5.5 端口 .....	309
3.14 文件读写命令 .....	162	4.6 图形类组件 .....	312
3.15 系统处理命令 .....	175	4.6.1 图片框 .....	312
3.15.1 了解剪辑板 .....	175	4.6.2 画板 .....	314
3.15.2 了解注册表 .....	175	4.6.3 颜色选择器 .....	335
3.15.3 系统处理命令 .....	175	4.7 分组类组件 .....	336
3.16 媒体播放命令 .....	191	4.7.1 分组框 .....	336
3.16.1 常见的音频格式 .....	191	4.7.2 外形框 .....	337
3.16.2 媒体播放命令 .....	191	4.7.3 选择框 .....	339
3.17 网络通信命令 .....	195	4.7.4 单选框 .....	342
3.17.1 了解IP地址 .....	195	4.7.5 选择夹 .....	345
3.17.2 了解域名与主机名 .....	195	4.8 位置控制类组件 .....	350





4.8.1 进度条 .....	350	4.11.2 标签 .....	384
4.8.2 滑块条 .....	351	4.11.3 表格 .....	392
4.8.3 横向滚动条 .....	354	4.11.4 打印机 .....	400
4.8.4 纵向滚动条 .....	355	4.11.5 影像框 .....	432
4.8.5 调节器 .....	357	4.12 数据库类组件 .....	434
4.9 网络与通信组件 .....	358	4.12.1 数据库提供者 .....	434
4.9.1 客户 .....	358	4.12.2 数据源 .....	435
4.9.2 服务器 .....	360	4.12.3 通用提供者 .....	465
4.9.3 数据报 .....	364	4.13 核心库内置数据类型 .....	465
4.9.4 超级链接框 .....	366	4.13.1 库内置数据类型的使用 .....	466
4.10 时间类组件 .....	369	4.13.2 字体 .....	466
4.10.1 时钟 .....	369	4.13.3 打印设置信息 .....	467
4.10.2 月历 .....	370	4.14 我的播放器（四） .....	470
4.10.3 日期框 .....	375	4.15 小结 .....	474
4.11 显示类组件 .....	378	4.16 习题 .....	474
4.11.1 编辑框 .....	378		

### 第三部分 易语言的数据库应用

第五章 易语言数据库的应用 .....	476	5.7.5 记录的添加与修改命令 .....	499
5.1 了解易语言数据库 .....	477	5.7.6 记录的删除命令 .....	502
5.2 创建易数据库 .....	477	5.7.7 复制记录与复制结构命令 .....	504
5.2.1 使用菜单创建易数据库 .....	478	5.7.8 数值统计类命令 .....	505
5.2.2 使用代码创建易数据库 .....	479	5.7.9 记录的查找 .....	507
5.3 为易数据库添加记录 .....	482	5.7.10 索引的创建与使用 .....	509
5.4 易数据库密码的设置 .....	484	5.8 我的播放器（五） .....	512
5.5 易数据库相关组件 .....	486	5.9 小结 .....	516
5.5.1 易数据库相关组件分类 .....	486	5.10 习题 .....	516
5.5.2 易数据库相关组件介绍 .....	486	第六章 外部数据库的应用 .....	517
5.5.3 易数据库组件关联方法 .....	488	6.1 外部数据库简介 .....	518
5.6 程序界面设计与组件关联 .....	489	6.1.1 外部数据库组件 .....	518
5.6.1 易数据库程序的界面设计 .....	489	6.1.2 ODBC与ADO .....	519
5.6.2 易数据库程序的组件关联 .....	490	6.1.3 Access数据库 .....	520
5.7 易数据库的操作命令 .....	490	6.2 SQL语言应用 .....	522
5.7.1 数据库的打开与关闭 .....	491	6.2.1 常用的SQL语句 .....	523
5.7.2 数据库指针的跳转命令 .....	493	6.2.2 定义数据库用户的权限 .....	524
5.7.3 记录读取命令 .....	494	6.2.3 定义表的结构 .....	524
5.7.4 数据源的常用命令 .....	496	6.2.4 数据检索 .....	526





6.3 外部数据库组件 .....	529	6.4.1 应用实例 .....	540
6.3.1 “外部数据提供者”组件 .....	529	6.4.2 “数据库连接”组件 .....	543
6.3.2 应用实例 .....	532	6.4.3 “记录集”组件 .....	546
6.3.3 “外部数据库”组件 .....	533	6.5 小结 .....	556
6.4 数据库连接和记录集 .....	540	6.6 习题 .....	556
 <b>第四部分 易语言高级应用</b>			
<b>第七章 DLL的应用 .....</b>	<b>558</b>	10.2.2 类的继承性 .....	593
7.1 了解DLL .....	559	10.2.3 类的多态性 .....	594
7.2 编写DLL .....	559	10.3 小结 .....	596
7.3 编译DLL .....	561	10.4 习题 .....	596
7.4 调用DLL .....	562	<b>第十一章 对象和COM对象 .....</b>	<b>597</b>
7.5 小结 .....	563	11.1 对象的应用 .....	598
7.6 习题 .....	563	11.1.1 对象型变量的定义 .....	598
<b>第八章 API的应用 .....</b>	<b>564</b>	11.1.2 “对象”数据类型 .....	599
8.1 了解API .....	565	11.1.3 “变体型”数据类型 .....	608
8.2 定义API .....	566	11.2 COM对象的应用 .....	612
8.3 调用API .....	570	11.3 小结 .....	614
8.3.1 调用系统API .....	570	11.4 习题 .....	614
8.3.2 调用非系统API .....	574	<b>第十二章 易模块的应用 .....</b>	<b>615</b>
8.4 应用实例 .....	576	12.1 了解易模块 .....	616
8.5 小结 .....	579	12.2 易模块的开发与编译 .....	616
8.6 习题 .....	579	12.2.1 易模块的开发 .....	616
<b>第九章 OCX组件与类型库 .....</b>	<b>580</b>	12.2.2 易模块的编译 .....	618
9.1 OCX组件 .....	581	12.3 易模块的引用方法 .....	619
9.1.1 OCX组件的安装 .....	581	12.4 易模块的应用实例 .....	620
9.1.2 OCX组件的使用 .....	583	12.5 小结 .....	624
9.2 类型库 .....	584	12.6 习题 .....	624
9.2.1 类型库的安装 .....	584	<b>第十三章 易语言向导 .....</b>	<b>625</b>
9.2.2 类型库的使用 .....	586	13.1 了解易语言向导 .....	626
9.3 小结 .....	587	13.2 易语言向导支持库 .....	626
9.4 习题 .....	588	13.3 易语言向导的编写 .....	635
<b>第十章 面向对象 .....</b>	<b>589</b>	13.4 易语言向导的使用方法 .....	639
10.1 了解面向对象 .....	590	13.5 小结 .....	640
10.1.1 类的概念 .....	590	13.6 习题 .....	640
10.1.2 类的创建 .....	591	<b>第十四章 程序调试 .....</b>	<b>641</b>
10.2 类的特性 .....	592	14.1 了解程序调试 .....	642
10.2.1 类的封装性 .....	592	14.2 运行调试 .....	642





14.2.1 预编译调试.....	642	15.2 易语言程序的编译 .....	651
14.2.2 运行中的调试.....	643	15.2.1 易语言5.X版本的编译 .....	651
14.3 调试命令 .....	646	15.2.2 易语言4.X版本的编译 .....	652
14.4 小结 .....	648	15.3 编译生成安装软件 .....	653
14.5 习题 .....	648	15.4 编译安装应用实例 .....	655
第十五章 程序的编译与发布.....	649	15.5 小结 .....	659
15.1 编译前的配置 .....	650	15.6 习题 .....	659

## 第五部分 易语言程序的解读和程序设计思路

第十六章 解读学校图书管理系统.....	661	17.1 游戏策划分析 .....	684
16.1 软件需求分析 .....	662	17.1.1 游戏策划文档实例.....	684
16.1.1 软件使用环境.....	662	17.1.2 小结 .....	687
16.1.2 功能需求分析.....	662	17.2 解读游戏——对对碰 .....	688
16.1.3 软件运行环境需求.....	663	17.2.1 解读游戏构架.....	688
16.2 解读学校图书管理系统 .....	663	17.2.2 解读游戏逻辑.....	689
16.2.1 试运行程序.....	663	17.2.3 解读特殊块功能的实现.....	696
16.2.2 解读程序.....	664	17.2.4 解读重绘画面.....	702
16.3 后记 .....	682	17.3 后记 .....	706
第十七章 解读游戏——对对碰 .....	683		



## 第一部分 易语言的基础知识

在第一部分，将学习使用易语言编程的各种基础知识。

包括易语言的概述，编程环境与输入法的讲解，易语言数据类型的分类和使用，运算符和表达式的介绍，变量、常量的定义和使用，资源的添加和使用，子程序的编写与调用。

通过这部分的学习，为今后的学习打下坚实的基础。





# 第一章 概述

## 本章目标

.....

在本章结束时，我们能够：

- 了解易语言与众不同的、独有的历史、特点、功能和配置
- 掌握易语言的安装、启动与新建程序的方法
- 熟悉易语言的工作界面、菜单命令等
- 了解软件设计流程
- 完成第一个易语言程序



## 1.1 易语言概述

### 1.1.1 易语言简介

易语言是中国人拥有自主知识产权的一门编程语言。是一款编译器，也是一套集成开发环境。综合来说，易语言是一个自成体系的软件开发平台。其中包含了用于开发软件产品的几乎所有重要部件：编程语言、编译器、调试器、类库（支持库/模块）、集成开发环境（IDE）及周边工具。

易语言最初是由吴涛创造的一门计算机程序开发语言。以其易学、易懂、易用、功能强大著称，是以中文作为程序代码表达的语言形式。早期版本的易语言命名为“E语言”，版本的发布可追溯至2000年9月。在2004年3月由大连大有房地产开发有限公司投资成立大连大有吴涛易语言软件开发有限公司，使易语言有了新的发展和进一步的完善。

易语言开发环境是建立在Windows平台上，支持全中文、全可视化编程操作，功能丰富且易学易用，可以满足国内各类计算机用户的需求，并可直接在Windows环境下开发Linux程序。

作为一款由国人自己研发的全中文编程语言，易语言自然融入了中华文化和中国人的习惯，用户不再需要按照国外的语言习惯、表达方式、甚至是思维方式，而是直接使用中文，按照国人自己的习惯去思考并编写程序。

易语言编程环境方便直观、快捷实用，不但可以支持程序代码全部用中文来编写，而且操作界面亦为全中文，因此即使用户根本不懂英文或者对英文了解很少也能够快速地进入计算机程序开发的大门，甚至初中或小学文化水平的人也可以较快地学会编写一些简单程序。

易语言虽然是中文编写代码，但并不是把现有的编程工具简单地进行表面汉化或封装而成，它拥有自己独立的高质量编译器，中文源代码被直接编译为目的机器的CPU指令。甚至其编译器所编译出来的可执行代码与操作系统平台无关，因此能够很方便地实现跨平台编程。目前易语言可同时支持Windows和Linux程序的开发，这使易语言移植到其他操作系统平台极其方便，使之不再依赖特定的操作系统环境，同时这也符合国家发展开发自主知识产权基础系统软件的战略部署，易语言编程环境不但本身是一个重要的基础系统软件，而且还可以为其他自主知识产权操作系统提供配套、适合的应用软件开发工具。

易语言是一个开放性产品，它可以很好地跟其他开发工具协作，如：易语言可以调用其他编程语言开发的支持库（借助易语言支持库开发包）、COM（组件对象模型）、OCX（对象链接和嵌入用户控件）、DLL（动态链接库）等，易语言开发的标准DLL也可供其他语言调用，易语言可以使用几乎所有大中小型数据库，甚至允许在代码中直接嵌入





X86机器指令代码以期实现比汇编语言更底层的应用，5.1版本后还将支持与其他编程语言共享静态库文件（.lib、.obj）。

## 1.1.2 易语言的发展史

易语言1.0版始于2000年9月，最初的易语言是吴涛个人研发并发布。2001年1月吴涛正式提出具有中国自主知识产权的编程语言“易语言”。总体的设计、架构及所有的关键技术全部由吴涛本人完成，易语言中设计的60多万行的源代码也均由吴涛一人编写。

2004年3月8日对于易语言来说是盛大而隆重的日子，因为在这一天，大连大有吴涛易语言软件开发有限公司成立了。2004年4月16日，易语言公司发布了“易语言3.6纪念版”，此版更新并增加了很多的支持库，并且首次包含了易语言帮助文档（ESDN），以此来纪念易语言公司的成立。

2004年7月，易语言3.7正式版发布，并以此3.7版为基础开发出了易语言的英文版和日文版，同时参加了第二届中国国际软件和信息服务交易会。凭借在软交会的精彩、独特、全新亮相，2004年《易语言汉语编程环境》取得了《计算机软件著作权登记证书》，同年9月13日《易语言汉语编程环境》通过了中国科学技术委员会中国软件评测中心的严格测试，取得了《科技项目鉴定测试报告》；同年9月14日《易语言汉语编程环境》通过了中华人民共和国科学技术部机械工业信息研究院的科技查新，取得《科技查新报告》；同年9月17日，由大连市科技局在北京召开易语言科学技术成果鉴定会，会议由中科院院士张效祥将军为主任，中科院院士高庆狮为副主任等11位专家组成了鉴定会专家组，最后经专家委员会讨论后一致通过成果鉴定。

2005年9月，易语言4.0正式版发布。易语言4.0版相对于以前的版本做了很大的改动，根据易语言公司报道此次升级涉及到76项内容。不但更新了原有支持库的内容，同时还增加了十几个新的支持库；将代码编辑方式改为以文本方式编辑，并且在易语言程序的配置方式、调试方式、语法格式上都做了较大的改进。

2010年2月，易语言5.0正式版发布。此次版本升级可以说是易语言的又一里程碑，程序由以往的独立编译升级为静态编译。静态编译后的易语言可执行程序（exe）和动态链接库（dll），运行时不再依赖任何支持库文件；文件尺寸更小（相对以前的独立编译）；PE结构更合理（取消了“易格式体”）；加载速度更快，而且有效解决了“病毒误报”和“易被脱壳”的难题。

易语言的静态编译版本正在改进和完善中，我们期待易语言更加辉煌时刻的到来。

## 1.1.3 易语言的特点

### 1.1.3.1 全中文支持

易语言实现了彻底中文化，易语言使用者在编程的时候，不再需要先了解英文和语



法，甚至是西方的思维模式，只需要用汉语和中文思维方式便能写出软件。用户可轻松学习编程，无需跨英语门槛，如图1-1所示。

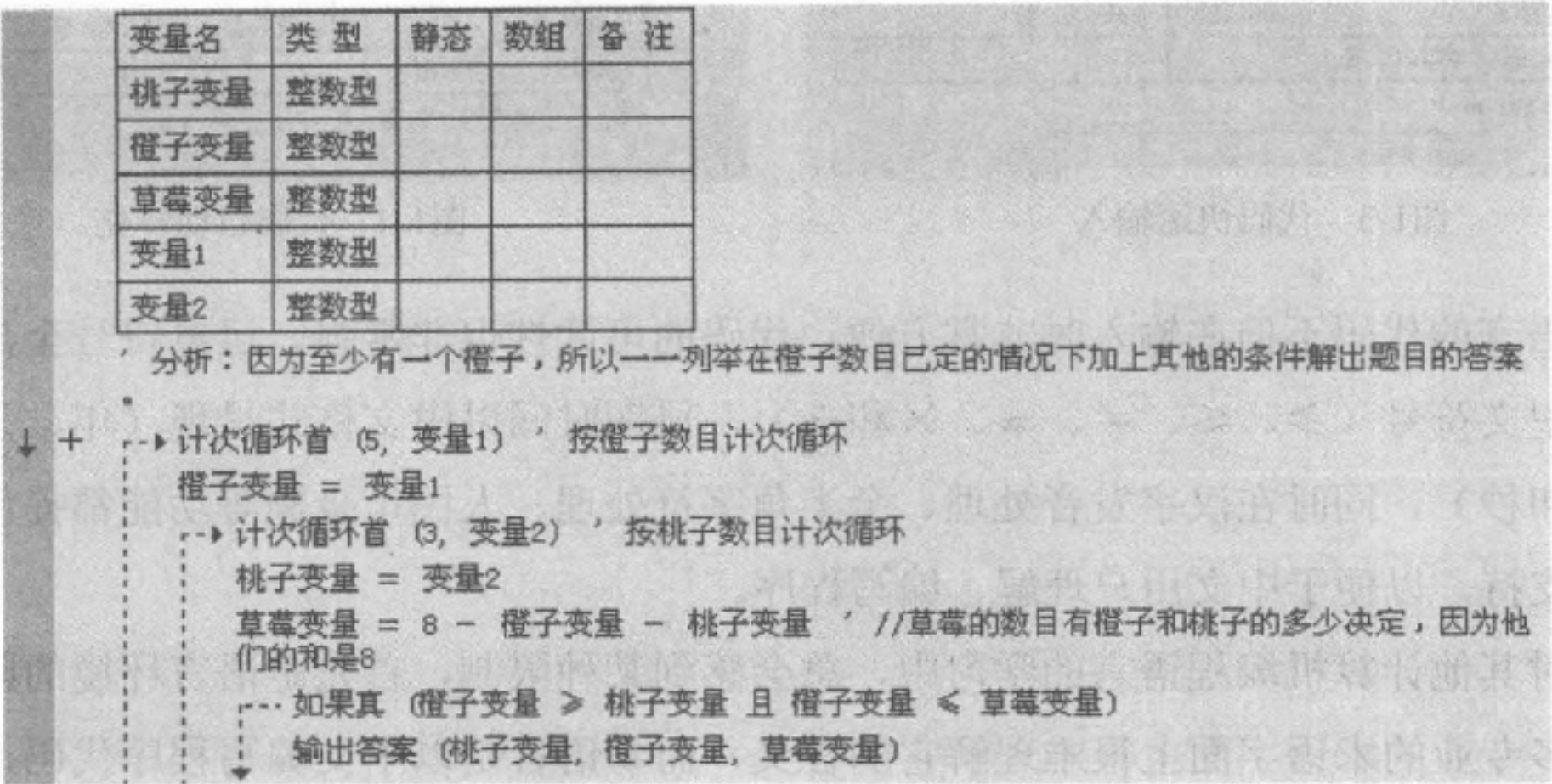


图1-1 全中文支持

1.1.3.2 全可视化编程

易语言支持界面设计的可视化和程序流程的可视化，如图1-2所示。易语言用户在编写程序的过程中，可以即时看到当前程序的运行流程及路线，有助于培养编程思路，提高解决编程问题的能力。对于学习编程语言的人来说，流程图是理顺程序设计思路、明确逻辑关系的最好办法，而易语言可以做到程序流程的“即输（输入）即画”，方便了用户，减少出错的可能性。

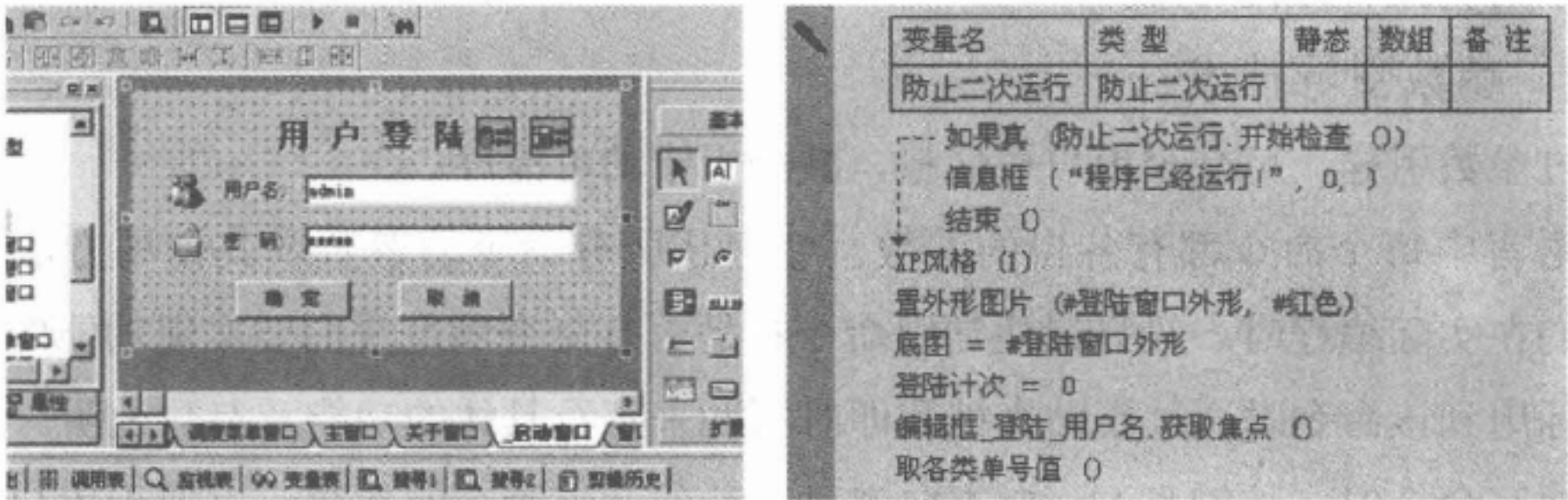


图1-2 界面设计可视化、程序流程可视

1.1.3.3 中文语句快速录入

易语言支持四种习惯的代码输入方式：首拼、全拼、双拼、英文，且均全面支持南方方言发音。使用这些输入方式能够极大地提高中文代码的输入速度，彻底解决了中文语句输入速度慢的问题。

这样一程序代码为：

信息框（编辑框1.内容，0，）

在易语言中可以输入，如图1-3所示。



该命令行的输入完成按回车键结束，系统会自动将首拼输入转换为标准代码样式，如图1-4所示。



图1-3 代码快速输入

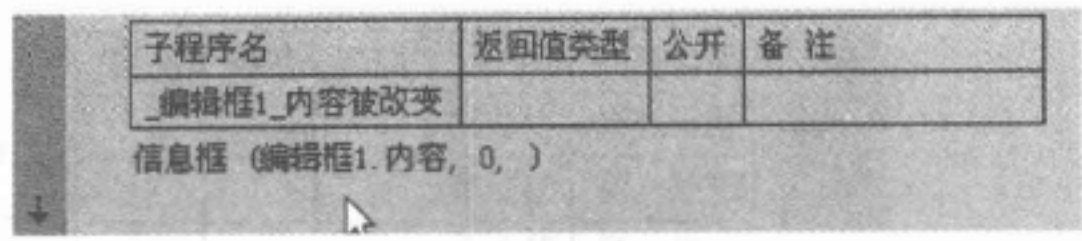


图1-4 代码自动转化

易语言的代码不但在输入时非常方便，代码的可读性也非常强。运算符号全部显示为对应的中文符号（ $\geq$ 、 $\leq$ 、 $\neq$ 、 $\approx$ 、 $\times$ 和 $\div$ ），日期时间以中文格式呈现（年、月、日、时、分和秒），同时在汉字发音处理、全半角字符处理、人民币金额等功能都提供了相当完美的支持，以便于中文用户理解、编写程序。

在对其他计算机编程语言的学习中，总会感到某种限制，首先是语言环境的限制，使得在很多专业的术语字面上很难理解它的含义，而易语言是以中文编写程序代码，符合中国人的语法习惯和逻辑思维，可以做到见文识义，更加适合中国人使用，并且在以后复查程序时可以非常直观地分析；使学习交流变得更加容易。

### 1.1.3.4 代码即文档

在易语言里面，任何人编写的源程序的样式和风格完全一致，便于工作中的交流与合作。这是其他编程语言所不具备而易语言独有的特性。无论您写代码时的格式多么不一致，大小写不分、空格乱空、句点不标准等，这些在编写完成后，系统会立即进行预编译处理，转换为标准样式风格。因此，任何人编写的易程序风格都是一致的，这极大地方便了工作中的交流、协作与维护。

### 1.1.3.5 参数引导技术

通过参数引导，可减少用户记忆量，减少出错的可能性。

易语言中每个命令都有各自的参数。要记住所有命令的参数及其使用方法是很难的。我们在实际编程时一般只记住常用命令和是否有命令满足我们要实现功能的需要，当需要实际用到该命令时通过帮助文档和即时帮助来查看具体的功能及参数说明。

当编写命令时，按键盘Alt+右方向键可自动展开该命令的参数，以便于用户书写程序代码，如图1-5所示。

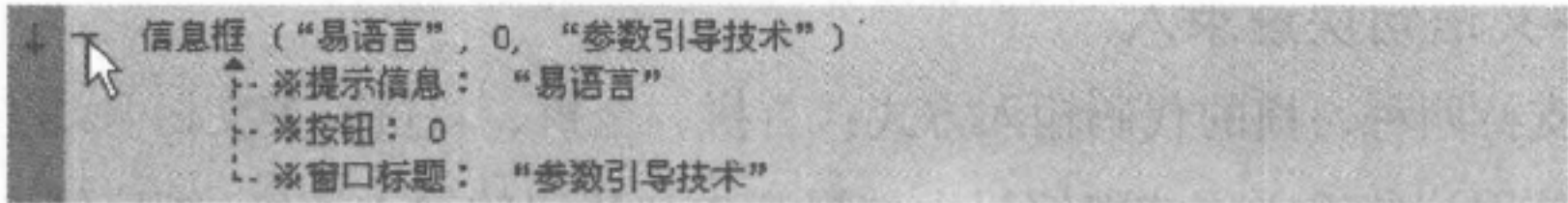


图1-5 参数引导技术

### 1.1.3.6 无定义类关键字

在易语言中，所有程序定义部分均采用表格填表方式，用户无需记忆此类关键字及其使用格式，如图1-6所示。



常量定义	常量名称	常量值	公开	备注
	用户习惯	"用户习惯"		
	姓名字段	1		
	年龄字段	2		
	性别字段	3		

数据类型定义	数据类型名	公开	备注
	点		点的位置
	成员名	类型	传址 数组 备注
	X轴	整数型	
	Y轴	整数型	

子程序定义	子程序名	返回值类型	公开	备注
	计算长度	文本型		
	参数名	类型	参考	可空 数组 备注
	长度	长整数型	✓	✓
变量定义	单位	整数型		✓
	变量名	类型	静态	数组 备注
	显示长度	文本型		
				1表示长度的单位为kb.默认为空代表0。

图1-6 定义表格

### 1.1.3.7 命令格式统一

易语言中，任何命令和关键字的调用格式完全统一，减少了编程人员编写程序的复杂度与记忆量，减少了出差错的可能性。易语言命令的调用格式统一为：

命令名（参数1，参数2，...，参数n），如图1-7所示。

"判断()" 命令	判断 (扩展名单选框.选中 = 真)	无论调用普通命令、流程命令或子程序，格式完全统一
"如果()" 命令	过滤文本 = 扩展名组合框.内容	
循环命令	如果 (过滤文本 ≠ "")	
普通命令	置等待鼠标 ()	
子程序调用	变量循环首 (全局_文件个数, 1, -1, 循环变量)	
	如果真 (匹配扩展名 (全局_旧文件名 [循环变量], 过滤文本) = 真)	
	如果真 (操作方式 = 1)	
	到循环尾 ()	
	启动窗口.显示列表框.删除表项 (循环变量 - 1)	
	删除成员 (全局_旧文件名, 循环变量, )	
	删除成员 (全局_旧文件名, 循环变量, )	
	更新信息 (过滤文本, 2)	
	全局_文件个数 = 全局_文件个数 - 1	
	变量循环尾 ()	
	信息框 ("扩展名不能空", #警告图标, "过滤设置")	
	返回 ()	
	判断 (包含内容单选框.选中 = 真)	
	过滤文本 = 包含内容编辑框.内容	

图1-7 命令格式统一

### 1.1.3.8 语法规则自动检查

在每行程序编写完成后按下回车键，系统会自动检查程序中的变量、常量或子程序的名称是否已被预先（声明）定义。如未被（声明）定义，系统会自动为其增加名称，以减少出错的可能性。在每行程序编写完成后按下回车键，系统会自动检查当前程序行的命令调用是否错误。如果发现错误，系统会及时给予提示，如图1-8所示。



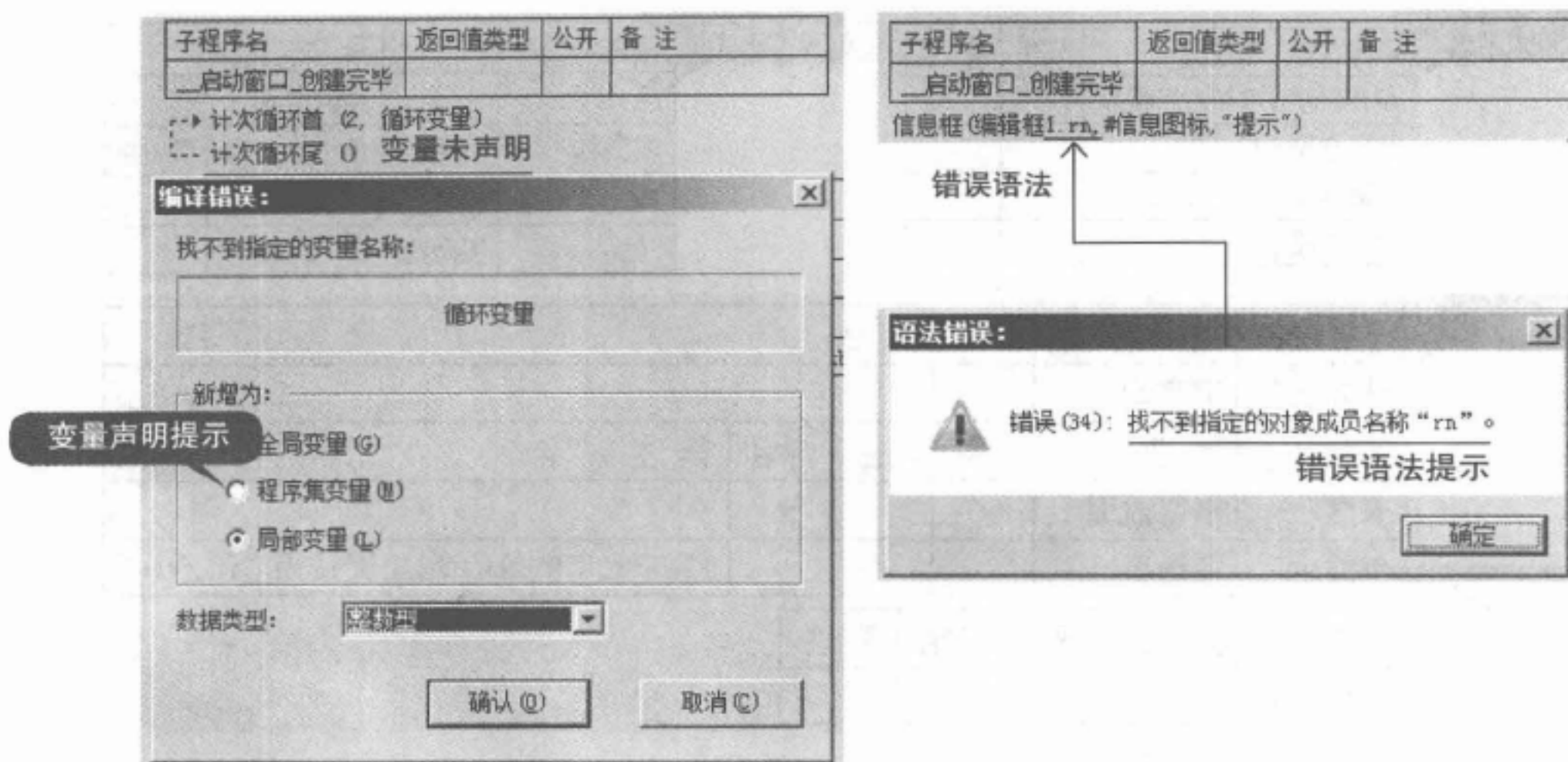


图1-8 语法格式自动检查

### 1.1.3.9 全程提示与帮助

易语言系统具有全程提示与帮助支持，用户随时随地都能查阅到相关信息。在进行任何操作的过程中，随时按F1帮助键，均能够在状态行上或提示夹中获得有关当前操作位置的详细信息。例如：用户将光标移动到某程序行上，然后按下F1键，马上就能够得到此程序行上所有命令的定义、参数、使用方法、所属支持库等信息，如图1-9所示。

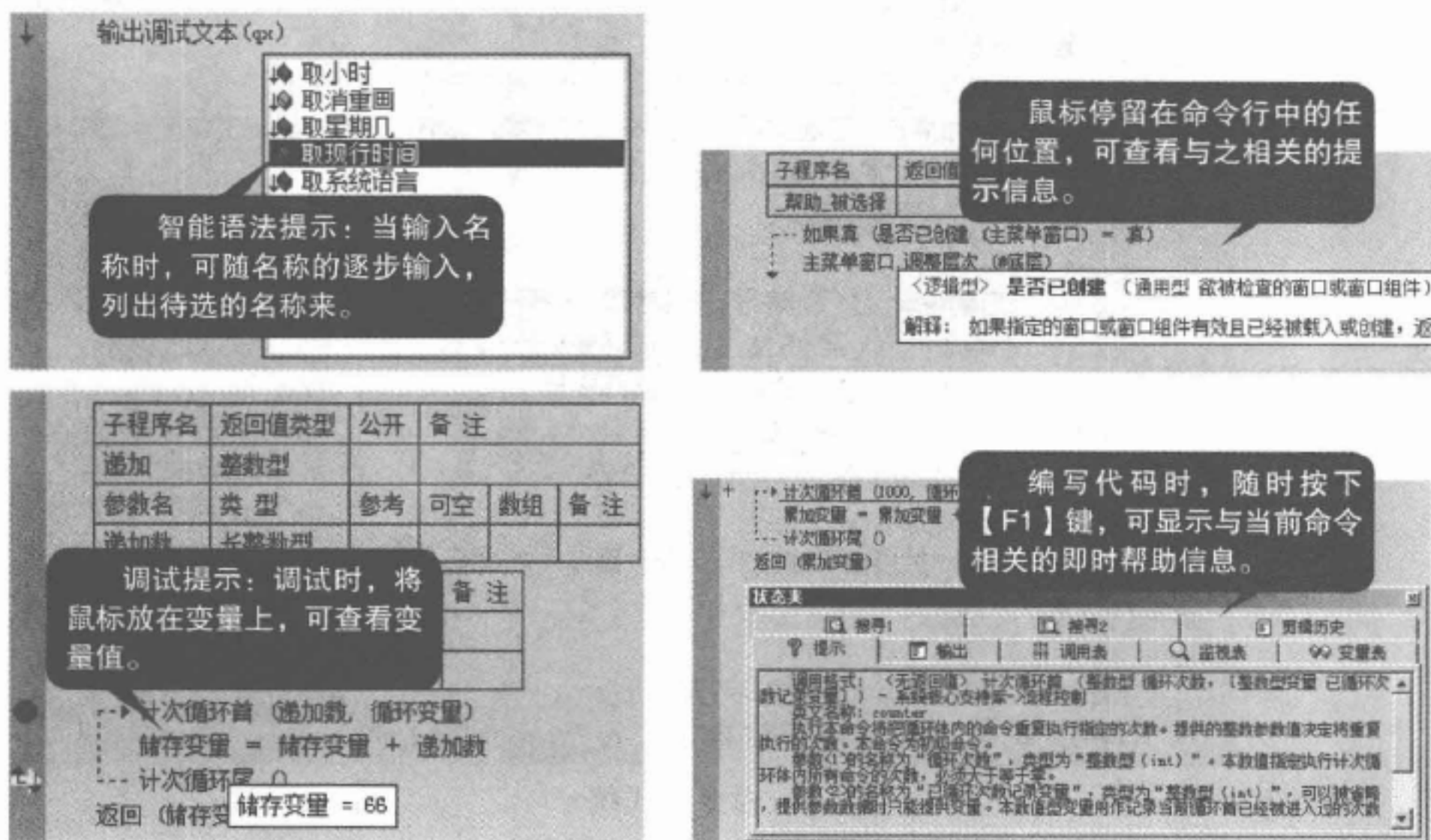


图1-9 全程提示与帮助

### 1.1.3.10 名称自动管理

易语言内置的自动名称管理器能够对用户所定义的各类名称进行跟踪管理。

如：假设程序中存在一个名为“提示信息”的子程序，并且在很多地方都调用了该子程序。现在用户根据需要为该子程序更改名称，在传统的编程语言中，用户需先更改该子



程序的名称，然后搜寻整个应用程序，找到使用了该子程序的地方，把名称进行更改。而在易语言中，用户只需更改该子程序的名称即可，系统会自动查找程序中所有使用了该子程序的地方，并快速地进行名称的更改，如图1-10所示。

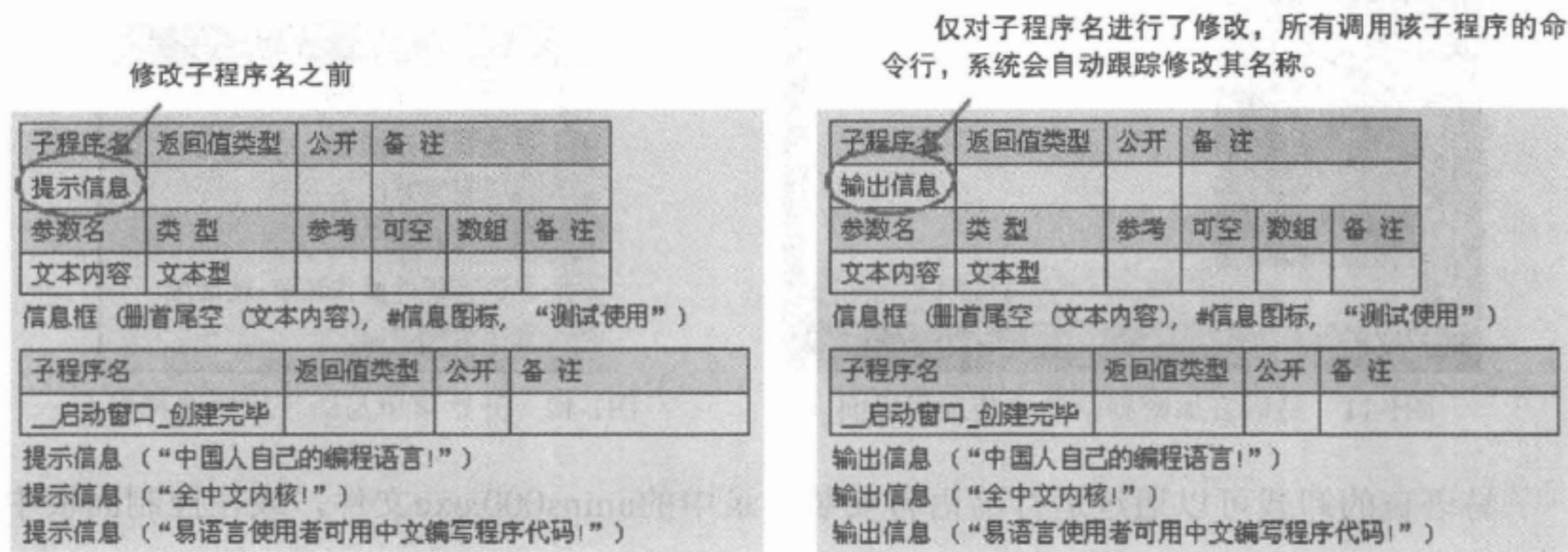


图1-10 名称自动管理

## 1.2 易语言的安装

易语言作为国人自主研发的全中文、全可视、跨平台编程环境，自然是希望更多的人、企业来自己开发真正适合自己、适合企业需求的软件。根据易语言官方论坛的显示，目前有超过 39 万的注册用户，每天上传的新代码有数百个。

在本书中将通过从易语言官方网站获取易语言最新安装程序，来给大家进一步讲解。

易语言官方主站：<http://www.dywt.com.cn>，<http://www.eyuyan.com>。

易语言官方论坛：<http://bbs.eyuyan.com>。

打开易语言主站首页，在网站栏目导航中选择——“产品下载”栏目，这里根据不同用户的需要提供了很多版本。对于初学者，建议选择下载易语言完整版。完整版中附带了帮助文档、多媒体教程和大量例程，可以方便读者学习和使用易语言。

易语言安装很简单，和很多软件的安装类似，安装过程中没有复杂的选项，双击安装程序即可开始安装，按照软件提示的操作，直到安装完成。（如果有光盘的用户也可以直接从光盘安装）。

这里值得注意的一点是，在安装结束时会出现图1-11所示的提示框，如果您已购买易语言企业加密狗版，那么在首次安装时应选择安装加密狗驱动程序，反之则可以不安装，然后点“完成”按钮来结束易语言的程序安装。

易语言安装完成，在桌面上和开始菜单中会出现易语言的快捷方式，如图1-12所示。点击快捷方式即可启动易语言。



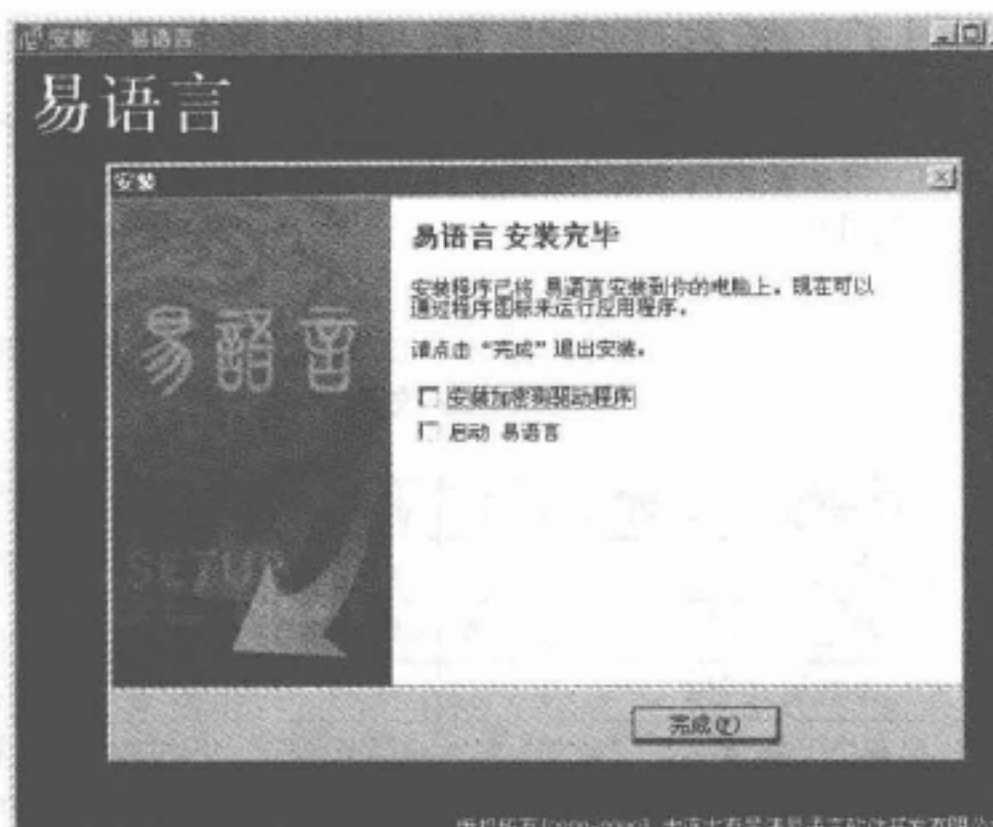


图1-11 易语言加密狗驱动安装选择界面

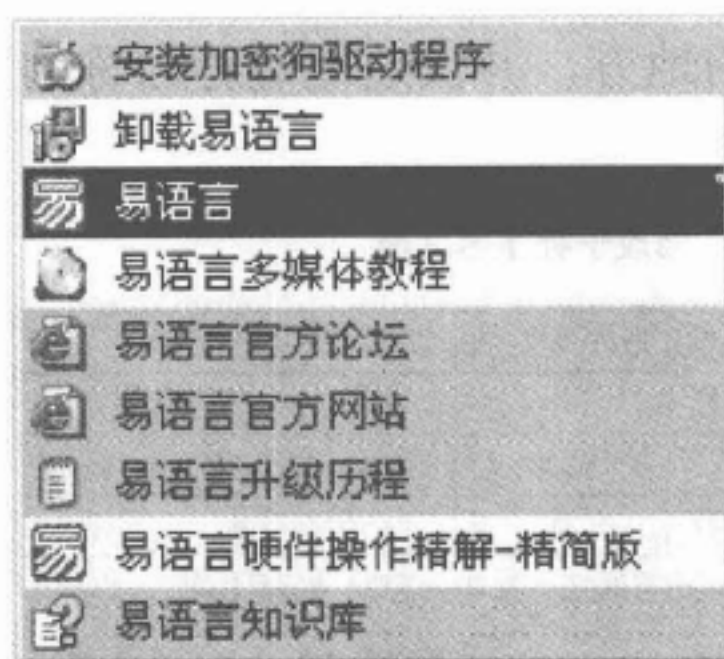


图1-12 开始菜单易语言程序选择界面

易语言的卸载可以通过运行易语言安装目录中的unins000.exe文件，或在控制面板中的“添加或删除程序”中进行卸载。

易语言安装后，安装目录中自动建立了几种子目录。下面了解一下几个比较重要的子目录的作用。

- (1) “clr”目录：存放易语言的各种颜色配置文件。
- (2) “ecom”目录：在导入过易模块以后，会产生此目录，用来存放导入的易模块(\*.ec文件)。
- (3) “help”目录：存放易语言的帮助文档。
- (4) “lib”目录：存放易语言的基本支持库和扩展支持库等重要文件。
- (5) “linux”目录：存放易语言在linux下使用的基本支持库和扩展支持库等重要文件。
- (6) “samples”目录：存放一些例程及源代码，供使用者学习参考。
- (7) “sdk”目录：存放易语言支持库开发包。
- (8) “setup”目录：存放易语言安装包文件。
- (9) “static\_lib”目录：存放易语言的静态基本支持库和静态扩展支持库等重要文件(5.0版本以上有)。
- (10) “tools”目录：提供了一些实用工具，如易之表、数据库语言转换器等。
- (11) “tutorial”目录：存放易语言多媒体教程。
- (12) “Wizard”目录：存放易向导可执行文件及易向导模板例程。

## 1.3 易语言的工作界面

### 1.3.1 易语言的界面

初次运行易语言，首先会弹出对话框，询问创建何种类型的易语言程序，如图1-13所示。



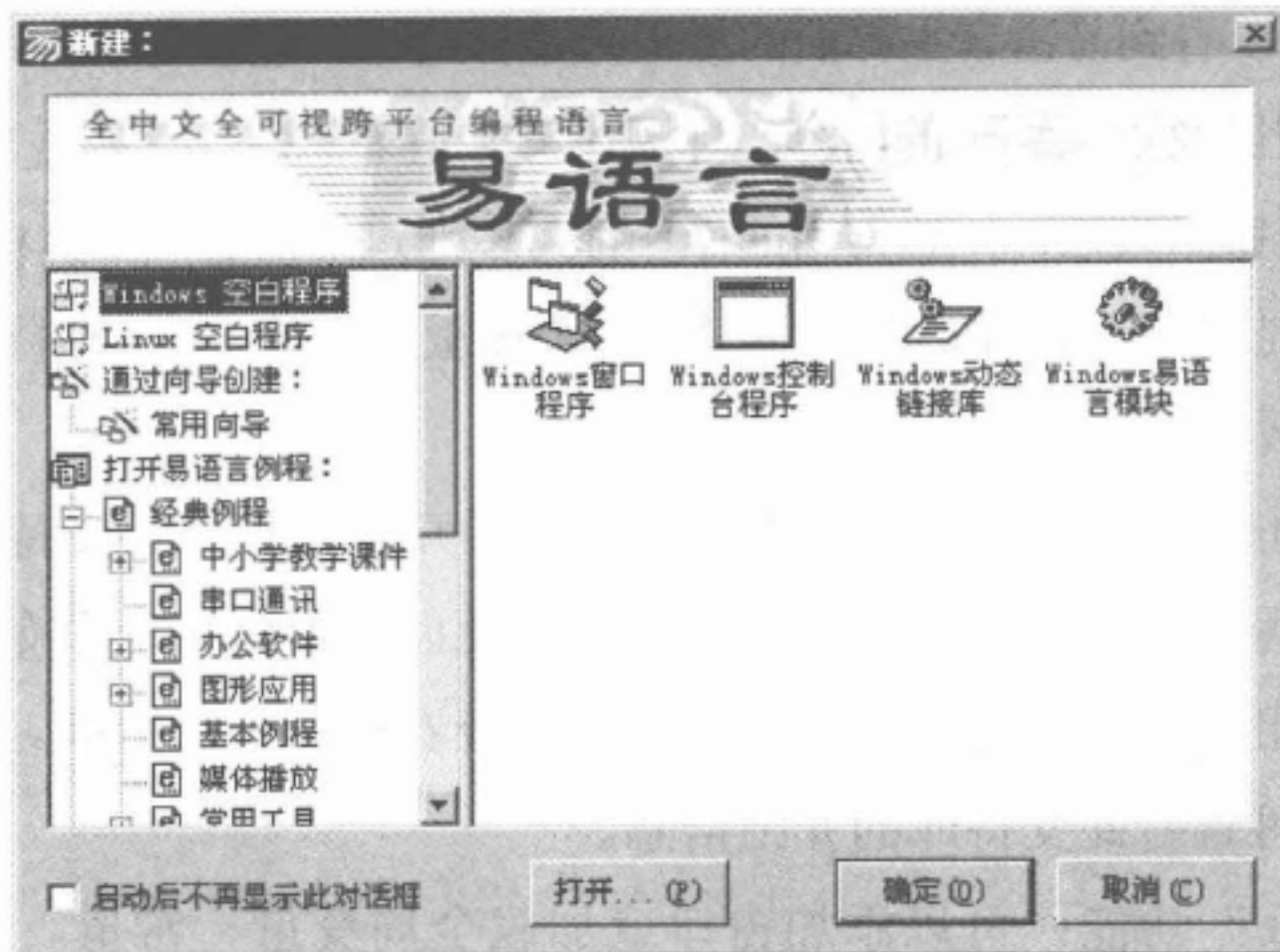


图1-13 易语言启动对话框

若打开易语言界面后未新建程序，也可通过菜单“程序”→“新建”来创建新的易语言程序，或点击窗口工具条中的新建按钮来新建易语言程序。

易语言可创建以下6种程序。

- (1) Windows窗口程序：支持在Windows下拥有窗口及组件等标准WIN32程序。
- (2) Windows控制台程序：Win32无窗口界面命令程序。
- (3) Windows动态链接库：可以生成DLL库文件。
- (4) Windows易语言模块：简称易模块，是经过初步编译后的代码集合，供其他易程序重复调用。

(5) Linux控制台程序：是支持Linux操作系统的无窗口命令程序。

(6) Linux易语言模块：是支持Linux操作系统且经过初步编译后的代码集合。

选择“Windows窗口程序”，点击“确定”按钮，会创建一个标准Windows窗口程序，并可以看到易语言的主界面，如图1-14所示。

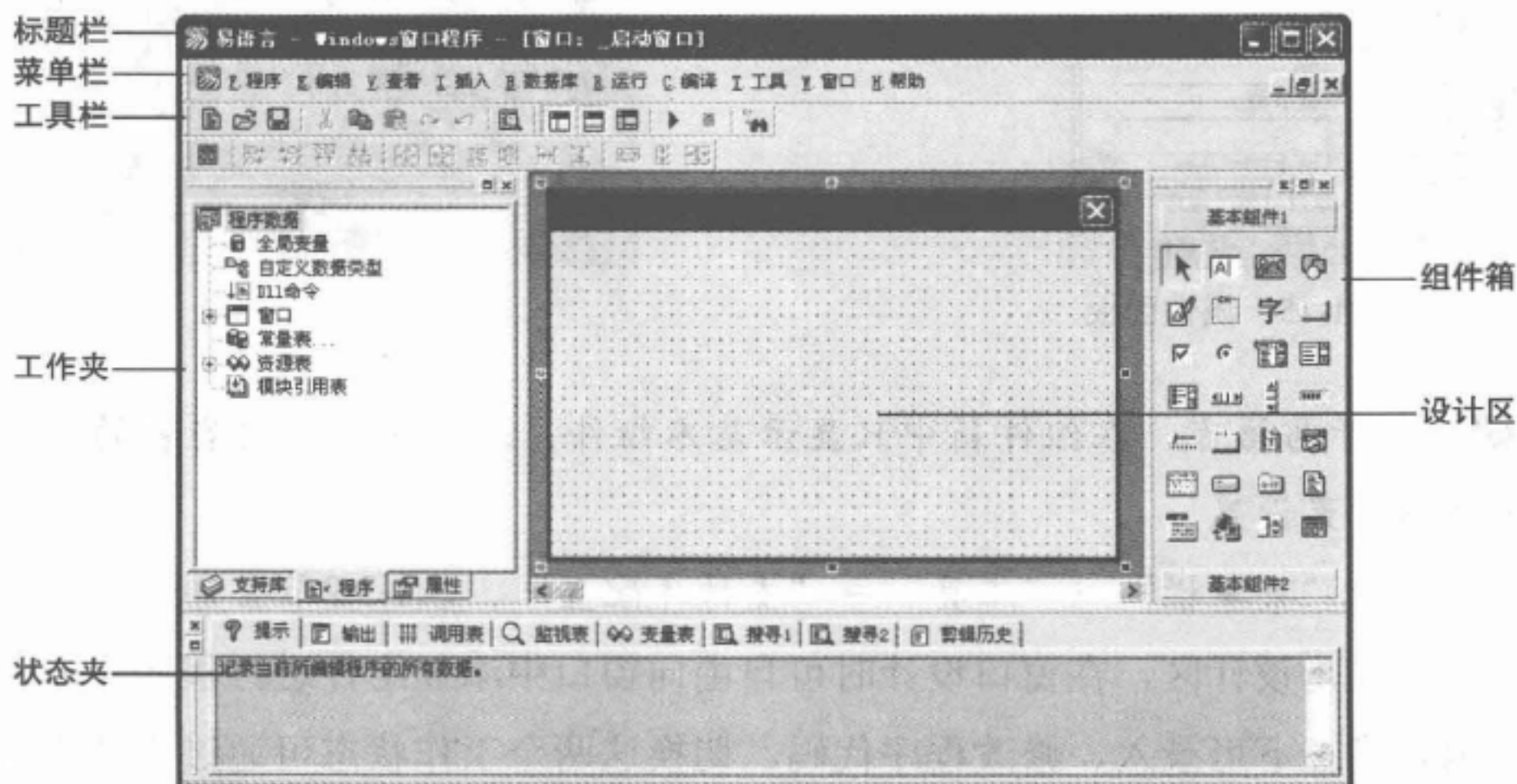


图1-14 易语言主界面



从图1-14中, 可以看到易语言主界面的最上方是标题栏, 显示易语言系统当前打开的程序名称, 当前程序类型以及正在编辑的内容。标题栏下方是菜单栏, 有易语言的常用菜单。菜单栏下方是快捷命令按钮工具栏, 一些常用的操作都可以通过点击这些工具栏中的按钮实现。

主界面的左边是易语言的工作夹, 包含有3个面板, 分别是“支持库面板”、“程序面板”和“属性面板”。

(1) “支持库面板”功能: 显示支持库列表, 展开后可查看各支持库提供的命令、数据类型等信息。在程序编辑状态下, 可以通过双击此面板中的某个命令, 将其直接填充到光标处。若有窗口组件的命令也可以在这个列表中查看该命令的用处。将光标移至某支持库根部, 按下F1后可查看此支持库的介绍信息。

(2) “程序面板”功能: 可以添加窗口或加载全局变量、常量、资源、DLL命令声明、自定义数据类型等。也可用来在程序各操作界面间进行切换, 例如可以直接找到某个创建的窗口或快速找到某个子程序。

(3) “属性面板”功能: 属性面板由属性表、组件列表、事件列表构成, 如图1-15所示。属性表可查看和更改已添加组件的属性; 组件列表列出所有组件并可快速选择所需组件; 事件列表可生成此组件的事件子程序。

图1-14中, 最右边是易语言的组件箱, 里面列出了易语言提供的所有组件, 如图1-16所示。分为四栏, “基本组件”栏显示的是易语言最基本常用组件, 即核心支持库内的组件。“扩展组件”栏显示的是扩展支持库内的组件。“外部组件”栏显示的是COM组件和OCX组件。“外部事件组件”栏显示的是COM事件组件。



图1-15 属性面板



图1-16 组件箱

**注解:** 打开易语言, 在组件箱中只显示基本组件, 这是由于初次安装的易语言还没有进行支持库配置设置。

**配置步骤:** “菜单栏” → “工具” → “支持库配置”, 在1.3.2节会详细讲解。

主界面中间是设计区, 在窗口设计时可自由向窗口中添加组件进行程序界面设计; 在程序代码编辑状态下可录入、修改程序代码。切换这两个工作状态可通过工作夹下“程序面板”中的“窗口程序集”和“窗口”来实现或者通过快速跳转夹来实现, 如图1-17所示。





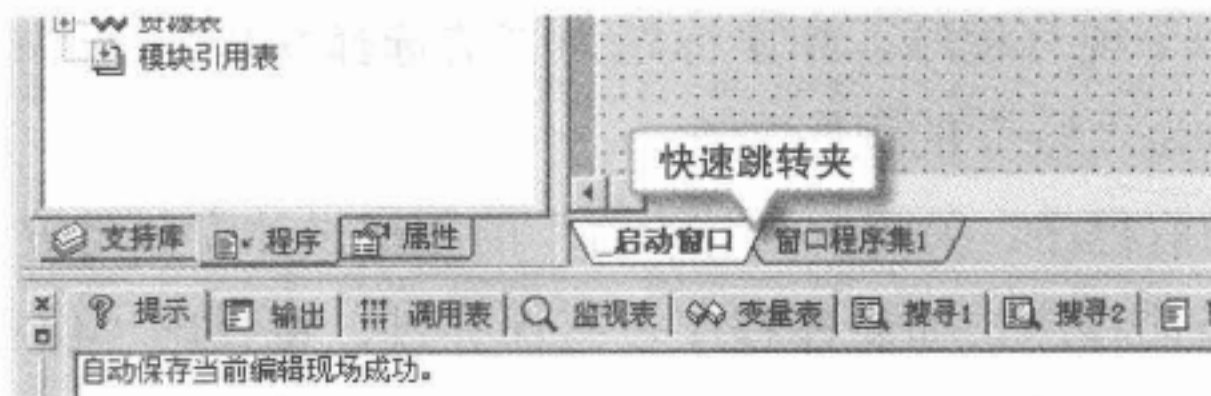


图1-17 快速跳转夹

图1-14中，最下方是易语言的状态夹，可以查看帮助信息，查看调试文本等。

### 1.3.2 易语言的菜单栏

下面以易语言默认创建的“Windows窗口程序”为例介绍各菜单项目的功能。

首先说明的是，菜单栏中各项目名称以及子项目名称前面带有下划线的字母是此项菜单的快捷键。只要项目上的文字未变灰，就可使用此项菜单的功能。例如：单击键盘上的Alt键，易语言系统菜单栏中的第一项“程序”被选中；单击菜单栏中任意一项名称前面的字母对应的键盘上的按键，此项菜单被弹出；最后单击子项目名称前面的字母，即可实现此菜单功能。

子项目名称后面的字母提示是此项功能的快捷方式。如：编辑菜单下的“复制”项后面提示“Ctrl+C”，功能是将所选内容复制到系统剪辑板中，方法就是同时按下键盘上的Ctrl键和C键或先按下Ctrl键然后再按下C键，而不需要弹出菜单。

在实际操作中，快捷键的使用会大大提高菜单功能的实现速度，减少鼠标的重复动作。

#### 1) 程序菜单

此菜单是程序操作相关的功能集合，如图1-18所示。

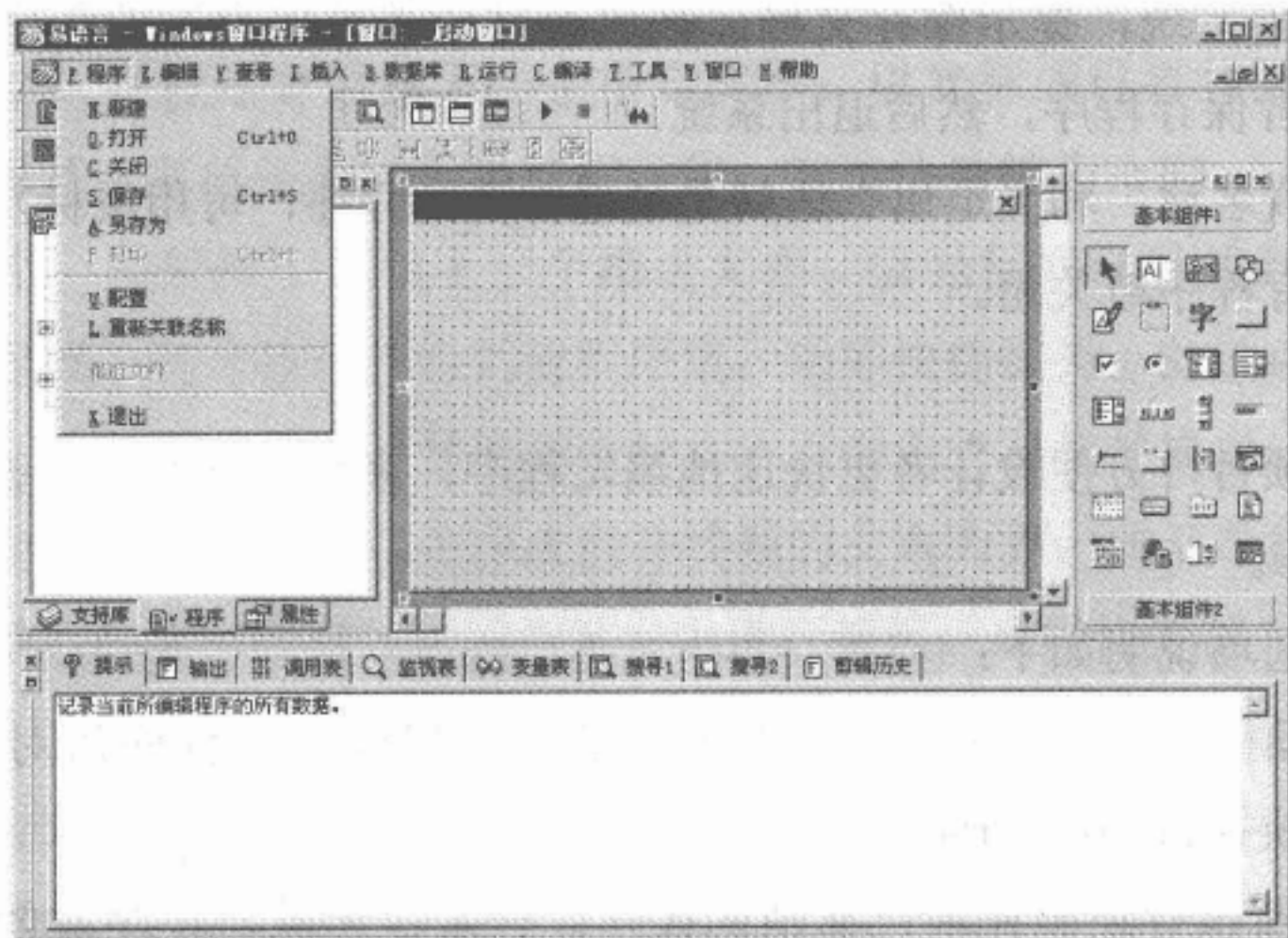


图1-18 程序菜单

程序菜单各选项说明如下：

- 新建：建立新程序。弹出标题为“新建：”的对话框，可以选择创建不同类型的程序。





- 打开：打开一个现有程序。弹出标题为“请选择易程序文件：”的打开文件对话框，选择后缀为“\*.e”的程序文件。
- 关闭：关闭当前程序。关闭后“易语言”的程序设计窗口将被置空。
- 保存：保存当前程序。如新建程序没有保存过，将弹出“保存为：”对话框，提示编辑者选择程序的保存位置和程序的名称，此后再次保存程序将默认这个保存位置，不会再弹出提示。
- 另存为：将当前程序以一个新文件名保存。将弹出“另存为：”对话框，提示编辑者选择程序的新的保存位置或输入程序的新的名称进行保存，同时将位置指向新保存的程序。
- 打印：打印当前编辑窗口中的源程序。使用打印机打印当前窗口中的源代码。
- 配置：配置本程序的环境及作者信息，如图1-19所示。

通过该对话框的设置，可以将程序名称、程序描述、程序备注、作者信息等信息保存在生成后的EXE、易模块、DLL文件中，当查看此文件的属性时，这些信息会显示出来。同时可以在“设置程序图标”中为程序设置图标。

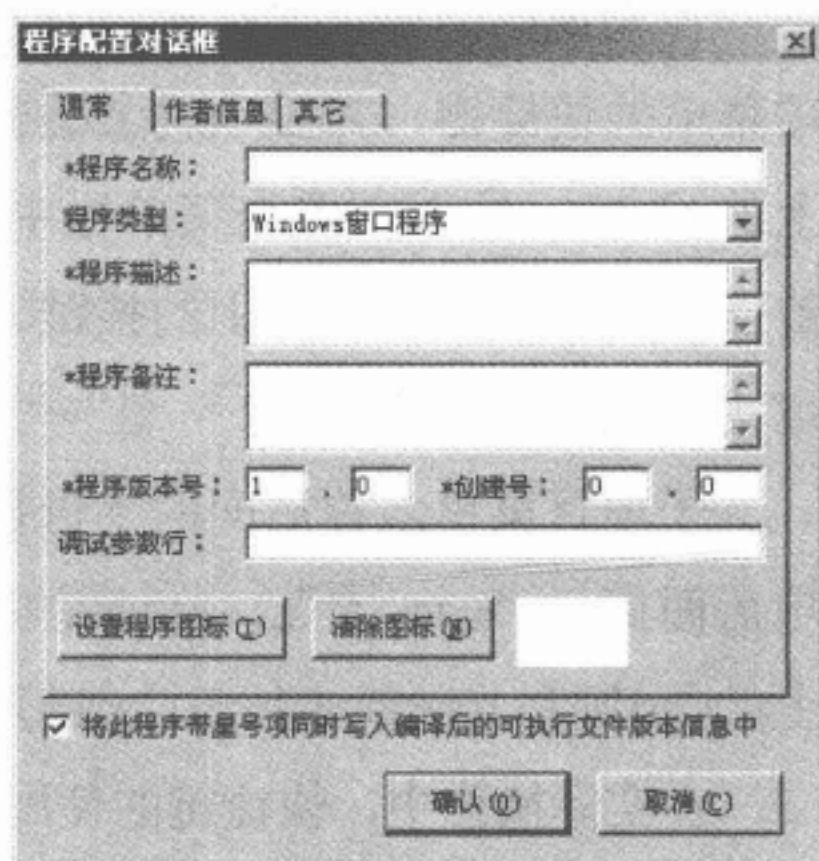


图1-19 程序配置对话框

- 重新关联名称：将程序中所有的名称关联起来，关联后的名称可以被系统内置名称管理器自动跟踪处理。
- 退出：退出系统，提示保存文档。被更改过或未被保存过的程序，将弹出信息框提示编辑者保存程序，然后退出系统。

在“重新关联名称”和“退出”之间是最近打开的程序，可用鼠标左键单击打开被选择程序，同时原有程序被关闭。

## 2) 编辑菜单

此菜单中的项目可方便设计者更快速地编辑程序，多数功能在代码设计区被激活时有效，如图1-20所示。

编辑菜单各选项说明如下：

- 撤消：撤消最后一步操作。一步步撤消自创建或打开程序后对程序的每一次修改。功能快捷键为：Ctrl+Z。
- 重复：重新执行还原先前已撤消的操作。一步步还原自程序被创建或打开后撤消的操作。功能快捷键为：Ctrl+Y。
- 复制：复制被选内容并将其置于系统剪辑板中。复制被选中代码或窗体、窗体组件到系统剪辑板，其原有内容不会改变。功能快捷键为：Ctrl+C。





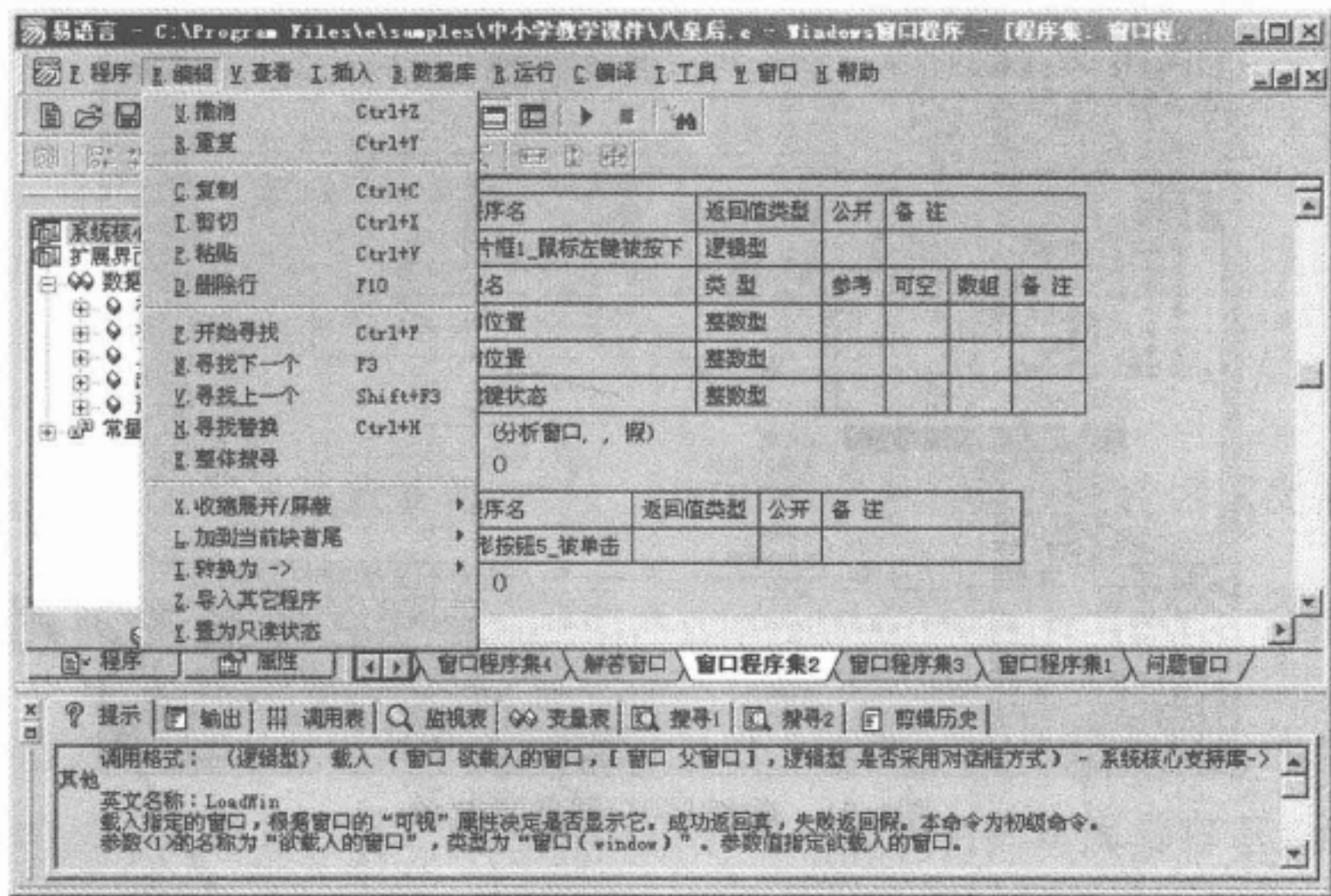


图1-20 编辑菜单

- 剪切：剪切被选内容并将其置于系统剪辑板中。相当于将被选中代码或窗体、窗体组件剪切到系统剪辑板中，其原有内容被删除。功能快捷键为：Ctrl+X。
- 粘贴：将系统剪辑板内容插入到当前位置。将系统剪辑板的内容插入到程序中。如果剪辑版内容是程序代码，需要在代码设计区中进行插入；如果内容是窗体组件，需要选中欲作为容器的窗口或窗口组件才能插入；如果是窗体，只需激活易语言系统，便可以将窗体插入到程序中。功能快捷键为：Ctrl+V。
- 删除行：删除当前所选择的区域或光标当前所在的行。功能快捷键为：F10。
- 开始寻找：开始在程序中寻找指定文本。弹出“寻找对话框”，请求输入被寻找的文本。其寻找范围为当前程序集。功能快捷键为：Ctrl+F。
- 寻找下一个：在程序中寻找下一个指定文本。以光标或已寻找到的文本为界，向代码下方寻找。其寻找范围为当前程序集。功能快捷键为：F3。
- 寻找上一个：在程序中寻找上一个指定文本。以光标或已寻找到的文本为界，向代码上方寻找。其寻找范围为当前程序集。功能快捷键为：Shift+F3。
- 寻找替换：在程序中寻找并替换指定的文本。弹出“寻找替换对话框”，提示输入被替换和替换成的文本。以光标或已寻找到的文本为界，向下寻找或替换文本，也可以将当前程序集中所有找到的指定文本进行替换。功能快捷键为：Ctrl+H。
- 整体搜寻：在程序中寻找指定文本并列出所有找到的项目。在全局中寻找指定文本，包括常量数据表、数据类型表等所有在代码设计区中以文本形式存在的指定项目。

**注解：** 以上功能遇到收缩的子程序时将跳过，不进入其内部寻找，忽略其中包括的指定文本。

- 收缩展开/屏蔽：将当前子程序或块内的所有语句收缩以“\*\*\* 缩略程序块 \*\*\*”显示，如图1-21所示。



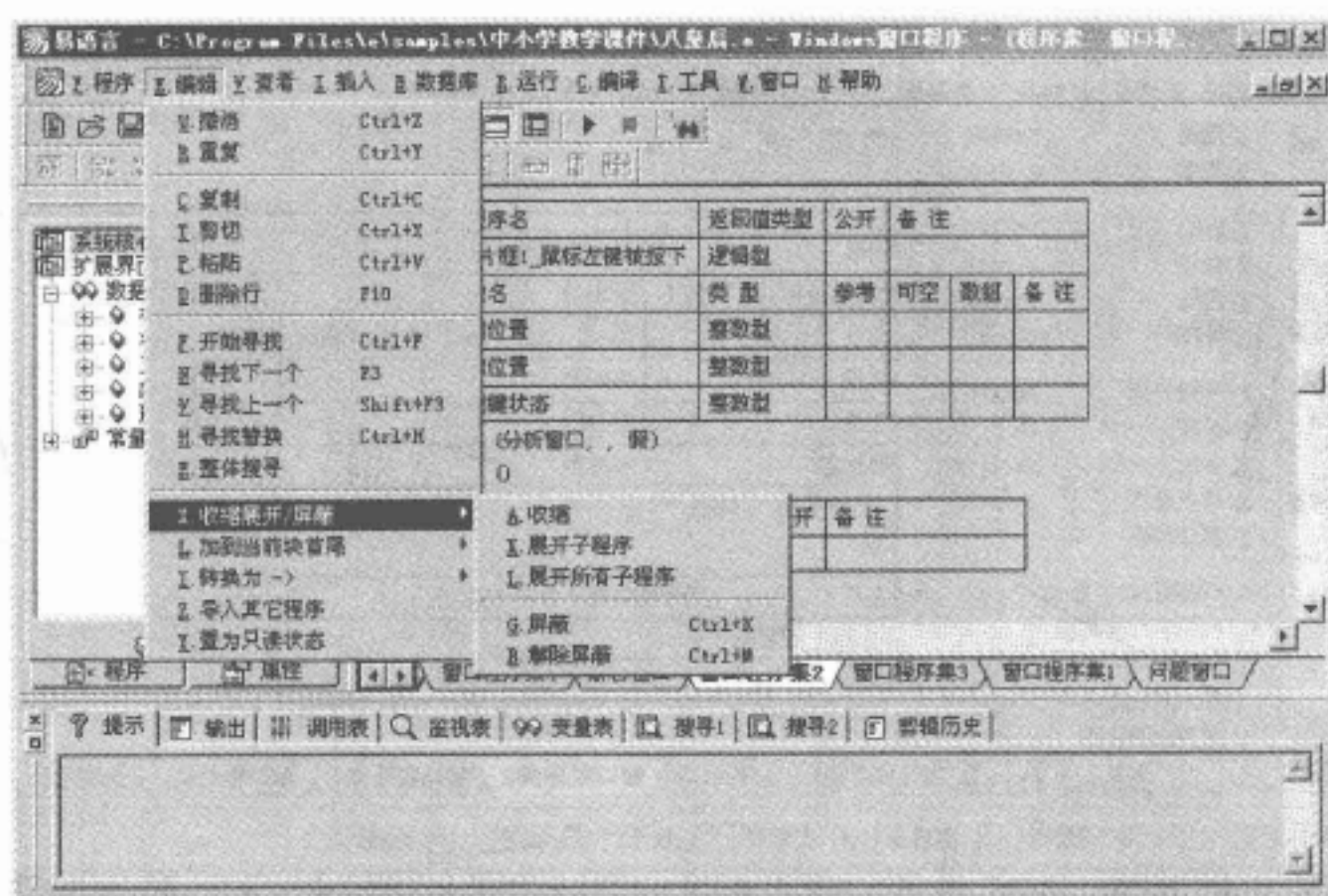


图1-21 收缩展开/屏蔽子菜单

- 收缩：将当前选中子程序或块内的所有语句收缩显示，如需打开请直接单击省略子程序或语句左边的加号图标。
- 展开子程序：将当前选中块内的所有被收缩子程序展开显示。
- 展开所有子程序：将当前程序中所有被收缩子程序展开显示。
- 屏蔽：屏蔽当前所选中的代码块。

把所选代码行或代码段设置为注释，使其在调试和运行程序时不被执行。功能快捷键为：Ctrl+K。

- 解除屏蔽：解除屏蔽当前所选中的代码块。

把注释行或被屏蔽的代码设置为可执行代码。功能快捷键为：Ctrl+M。

- 加到当前块首尾：如图1-22所示，将流程控制命令加入到当前所定义程序块的首部和尾部。添加一个新的流程控制命令，并将被选择代码块放到此命令中。要使菜单功能有效，选择代码块的方法是：选择两行或两行以上的单行代码；选择一个或多个分支流程控制命令，必须把流程线外的一行选中。

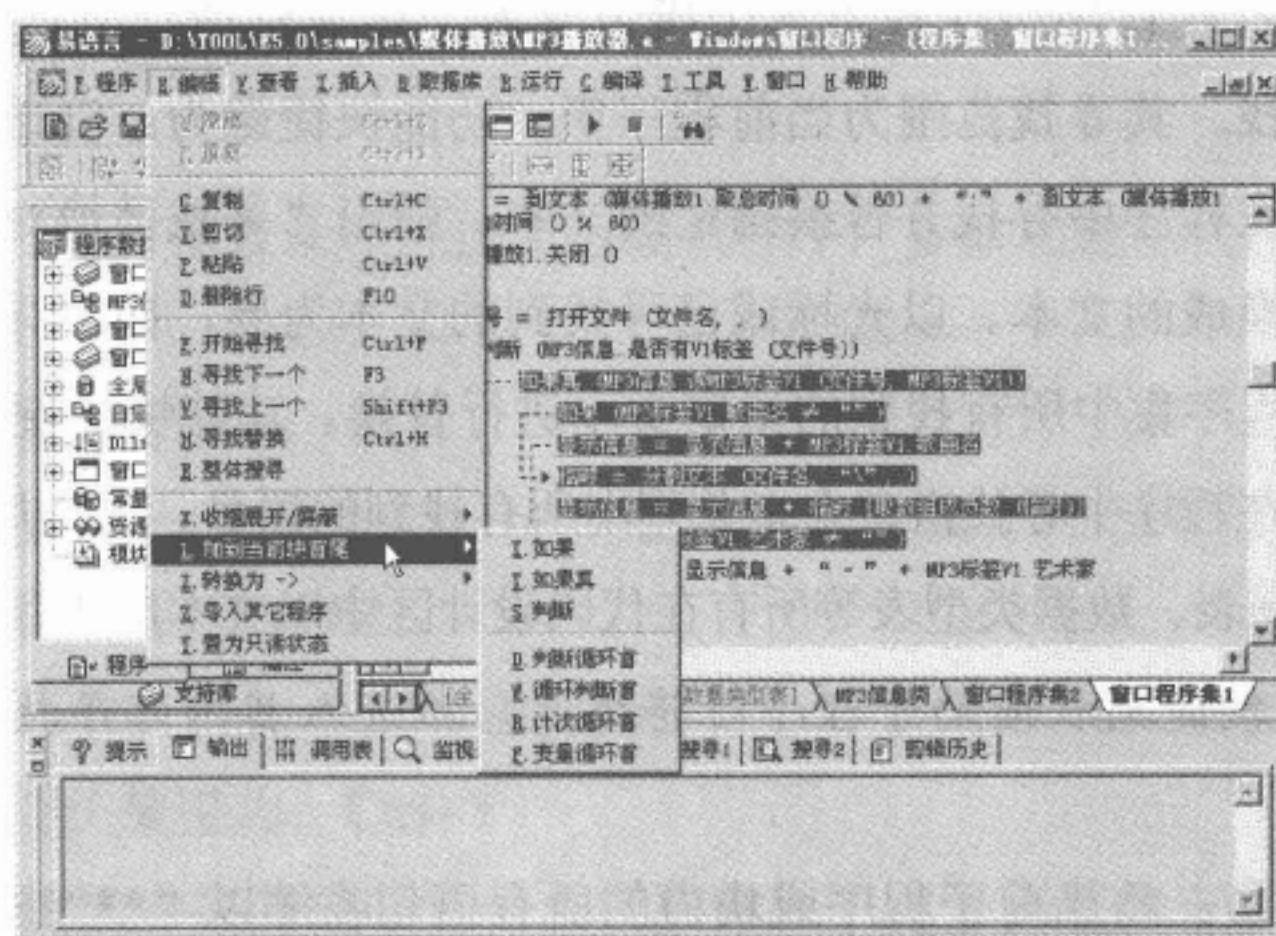


图1-22 加到当前块首尾子菜单



- 转换为->: 如图1-23所示, 将当前流程控制命令转换为别的流程控制命令。把选中的流程控制命令转换为别的流程控制命令, 它们之间可以互相转换。但要注意的是, 在转换过程中, 原来的程序流向有可能发生改变。

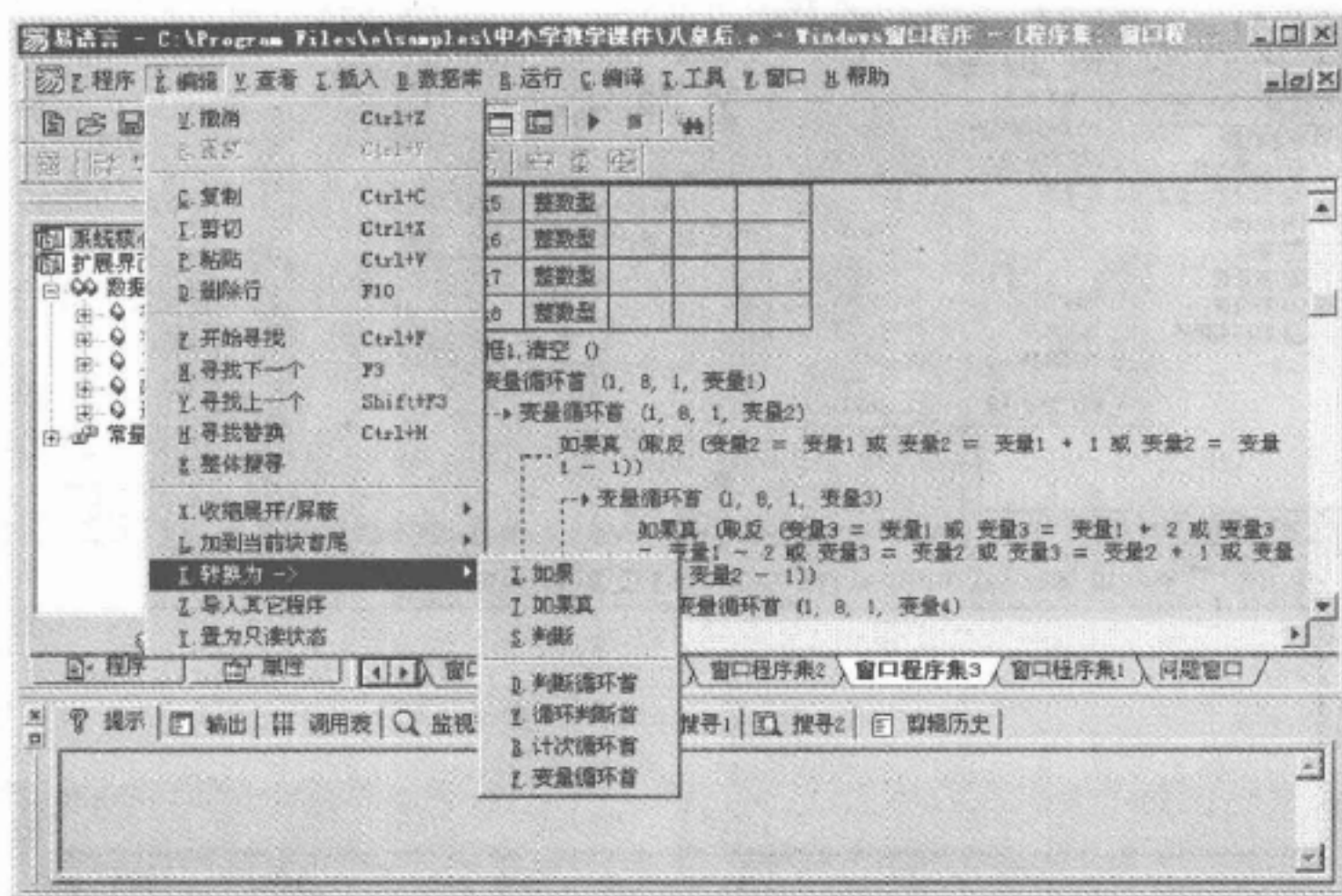


图1-23 转换为子菜单

- 导入其它程序: 将其他易程序中的内容全部导入到本程序中。打开标题为“请选择易程序文件:”对话框, 选择程序文件插入到当前程序中。被导入程序的“\_启动窗口”以及其他与当前程序重复的程序集名称、窗口名称后面将按顺序被加入数字加以区别。
- 置为只读状态: 设置为只读状态后将不允许所有修改操作的发生。

### 3) 查看菜单

查看菜单包含以下几类功能: 控制工具条的显示; 跳转到程序内各种资源的定义处; 进行代码快速定位; 快速查看某窗口运行时的显示效果, 如图1-24所示。

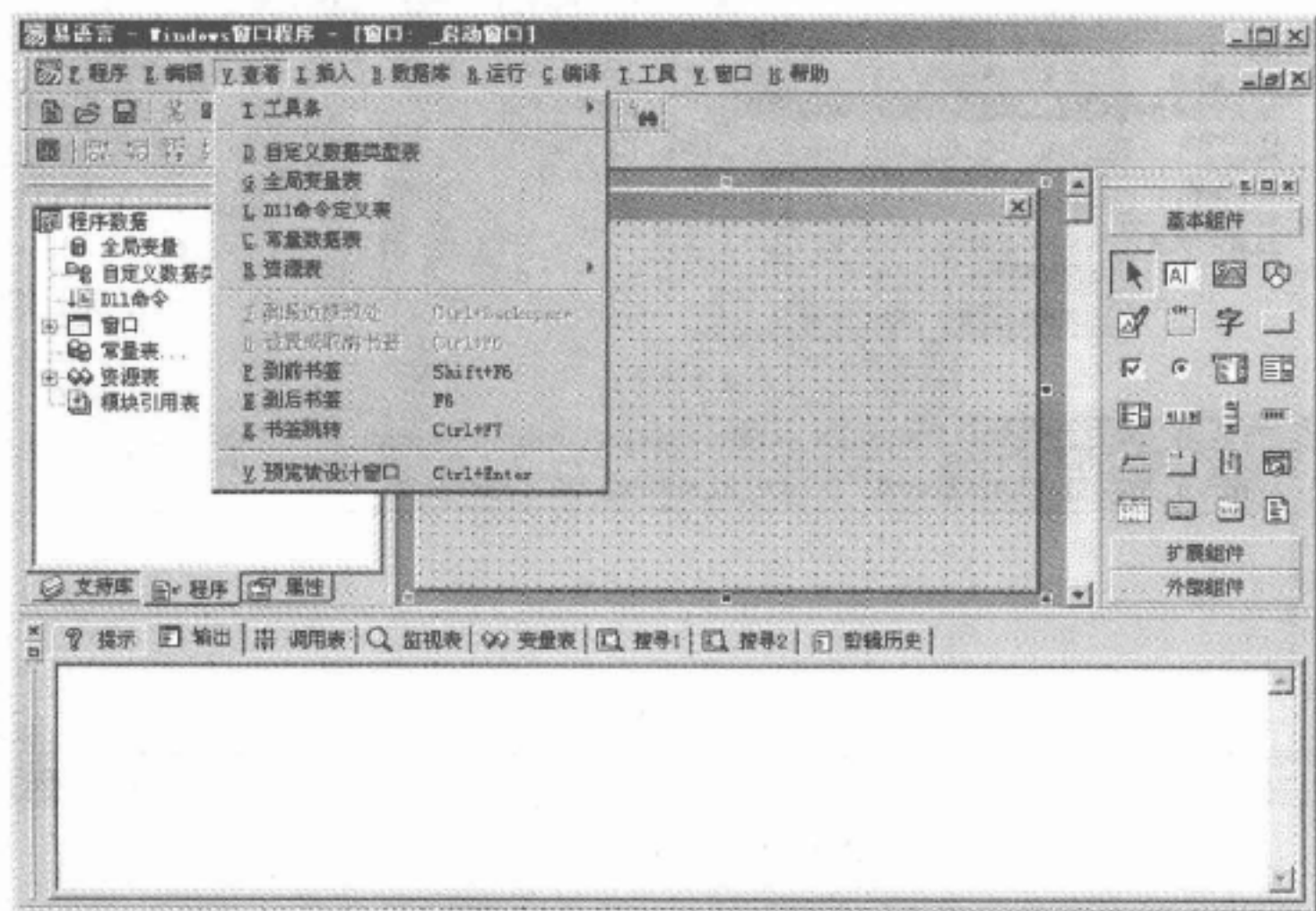


图1-24 查看菜单



- **工具条：**在易语言界面上显示和隐藏各功能工具，如图1-25所示。

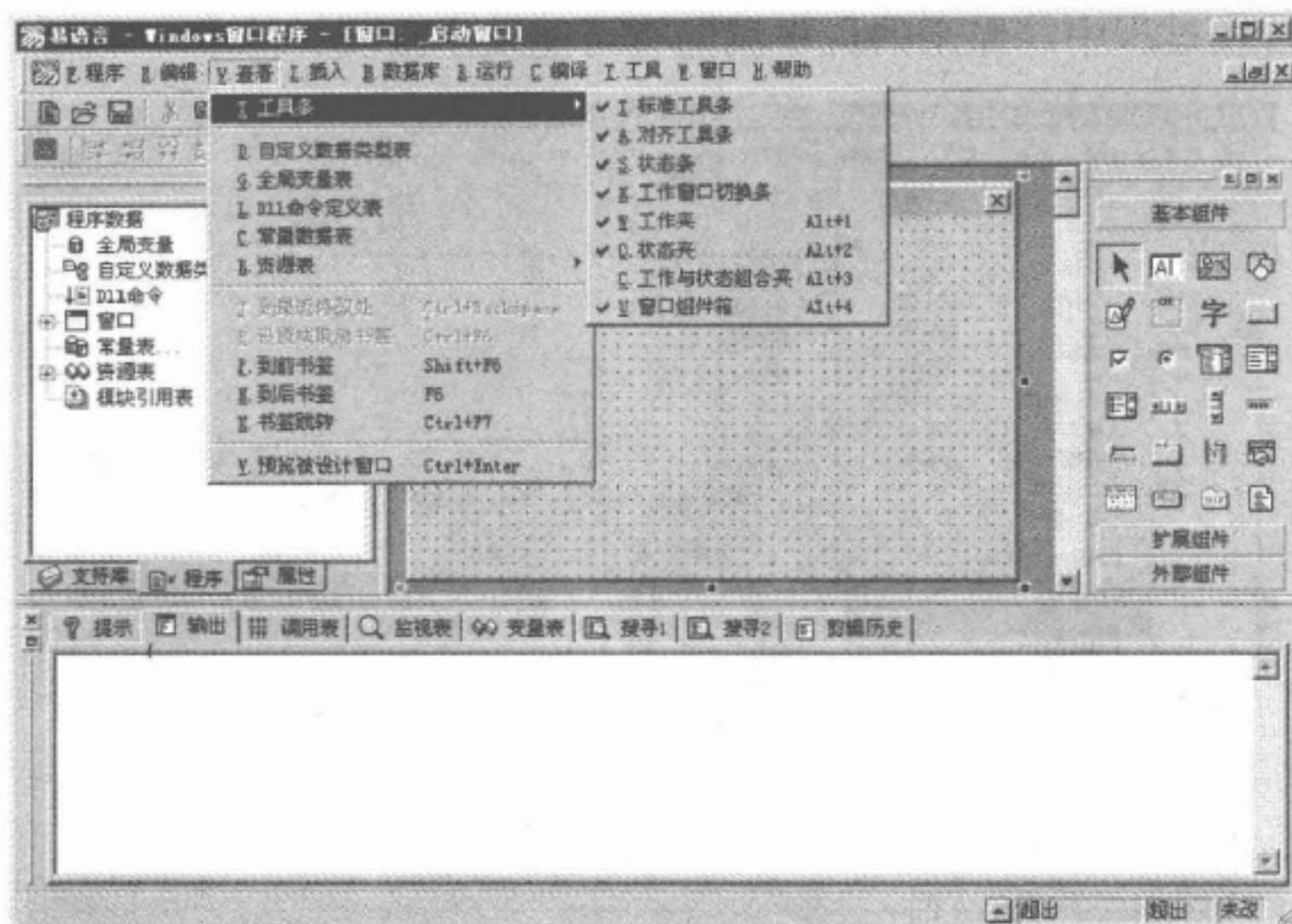


图1-25 工具条子菜单

- 自定义数据类型表：跳转到自定义数据类型定义表处。
- 全局变量表：跳转到全局变量定义表处。
- Dll命令定义表：跳转到Dll命令定义表处。
- 常量数据表：跳转到常量数据定义表处。
- 资源表：跳转到资源表处，与工作夹中程序面板中对应项目功能相同，如图1-26所示。

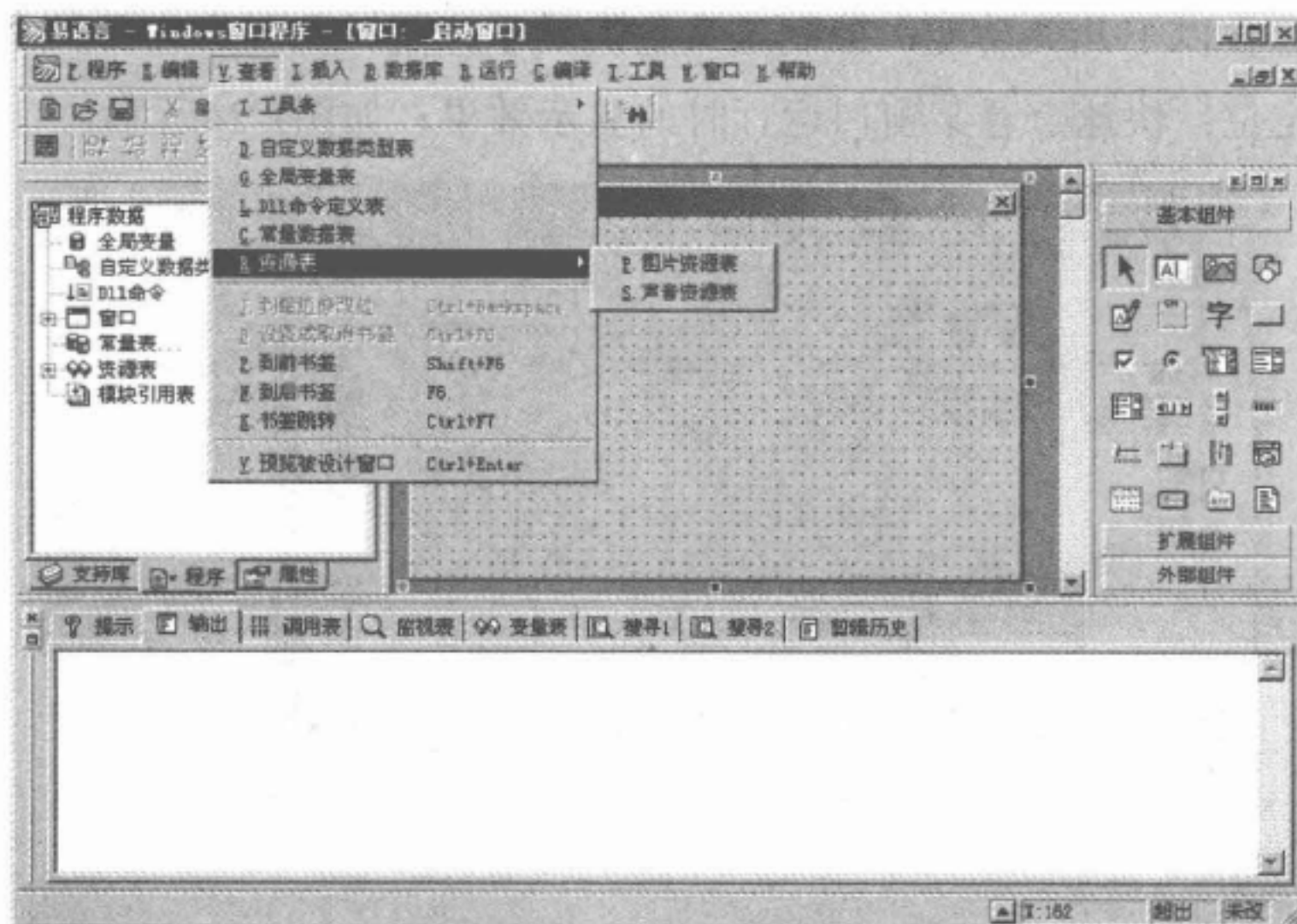


图1-26 资源表子菜单

- 到最近修改处：跳转到最近修改的位置。功能快捷键为：Ctrl+BackSpace（退格键）。



- 设置或取消书签：设置或取消当前位置处的书签标志。功能快捷键为：Ctrl+F6。
- 到前书签：跳转到前面的最近书签位置。功能快捷键为：Shift+F6。
- 到后书签：跳转到后面的最近书签位置。功能快捷键为：F6。
- 书签跳转：跳转到指定书签处。功能快捷键为：Ctrl+F7。
- 预览被设计窗口：预览现行窗口，按Esc键退出预览。预览时为程序编写的代码将不会被执行。功能快捷键为：Ctrl+Enter（回车键）。

#### 4) 插入菜单

将所选项目自动插入到对应的设计区中，如图1-27所示，由设计者按系统给定的格式填写代码。

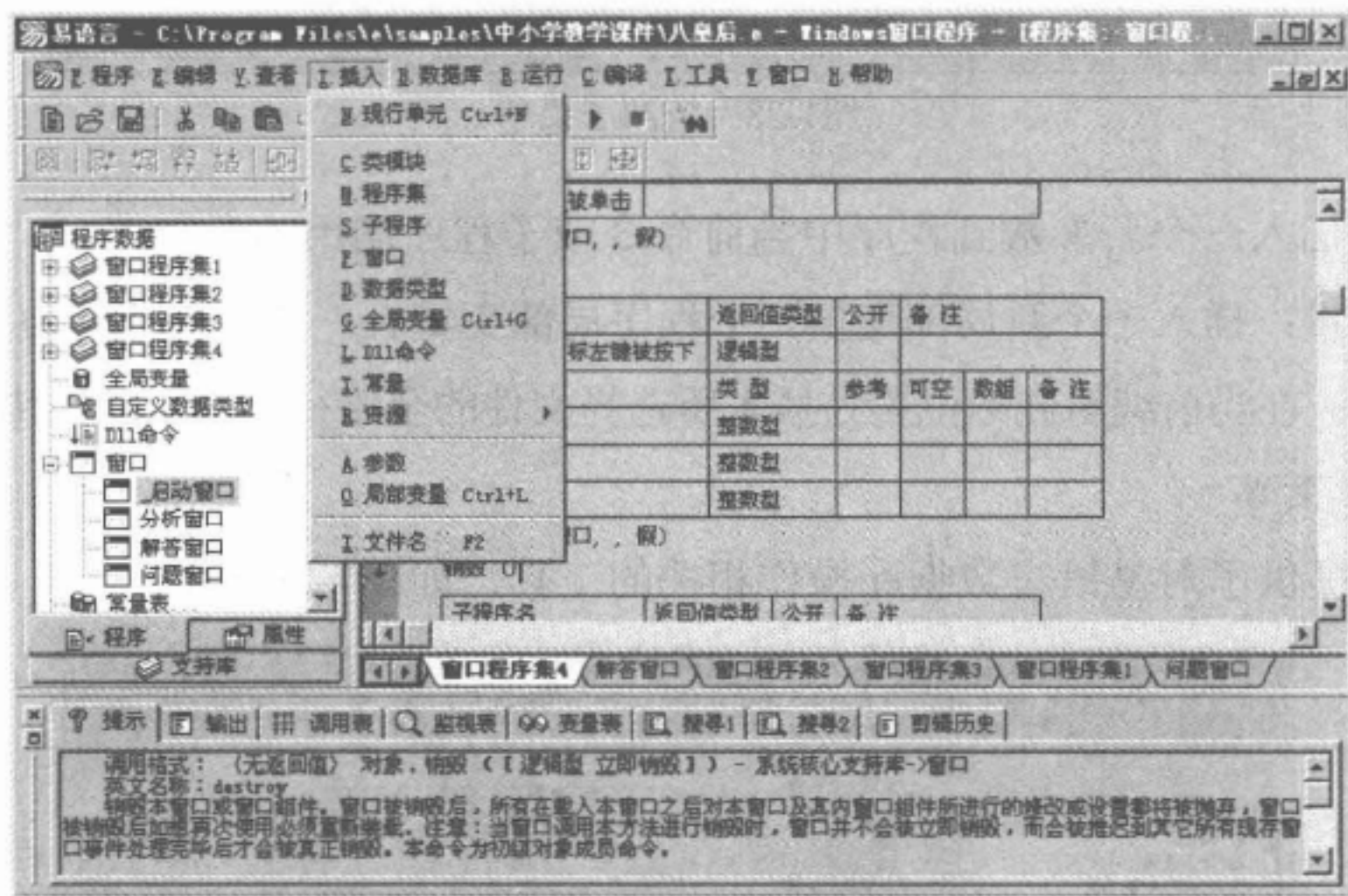


图1-27 插入菜单

插入菜单各选项说明如下：

- 现行单元：根据现行编辑窗口的性质插入一个新子程序/数据类型/全局变量/Dll命令/常量/资源到当前位置。功能快捷键为：Ctrl+N。
- 类模块：插入一个新类模块。
- 程序集：插入一个新程序集。
- 子程序：插入一个新的子程序到当前位置的后面。
- 窗口：插入一个新窗口。
- 数据类型：插入一个新的自定义数据类型到数据类型表。
- 全局变量：插入一个新的全局变量到全局变量表。功能快捷键为：Ctrl+G。
- Dll命令：插入一个新的Dll命令到Dll命令表。
- 常量：插入一个新的常量到常量数据表。
- 资源：向资源表中添加数据资源。如：文本文件、声音图片文件以及其他类型文件，如图1-28所示。



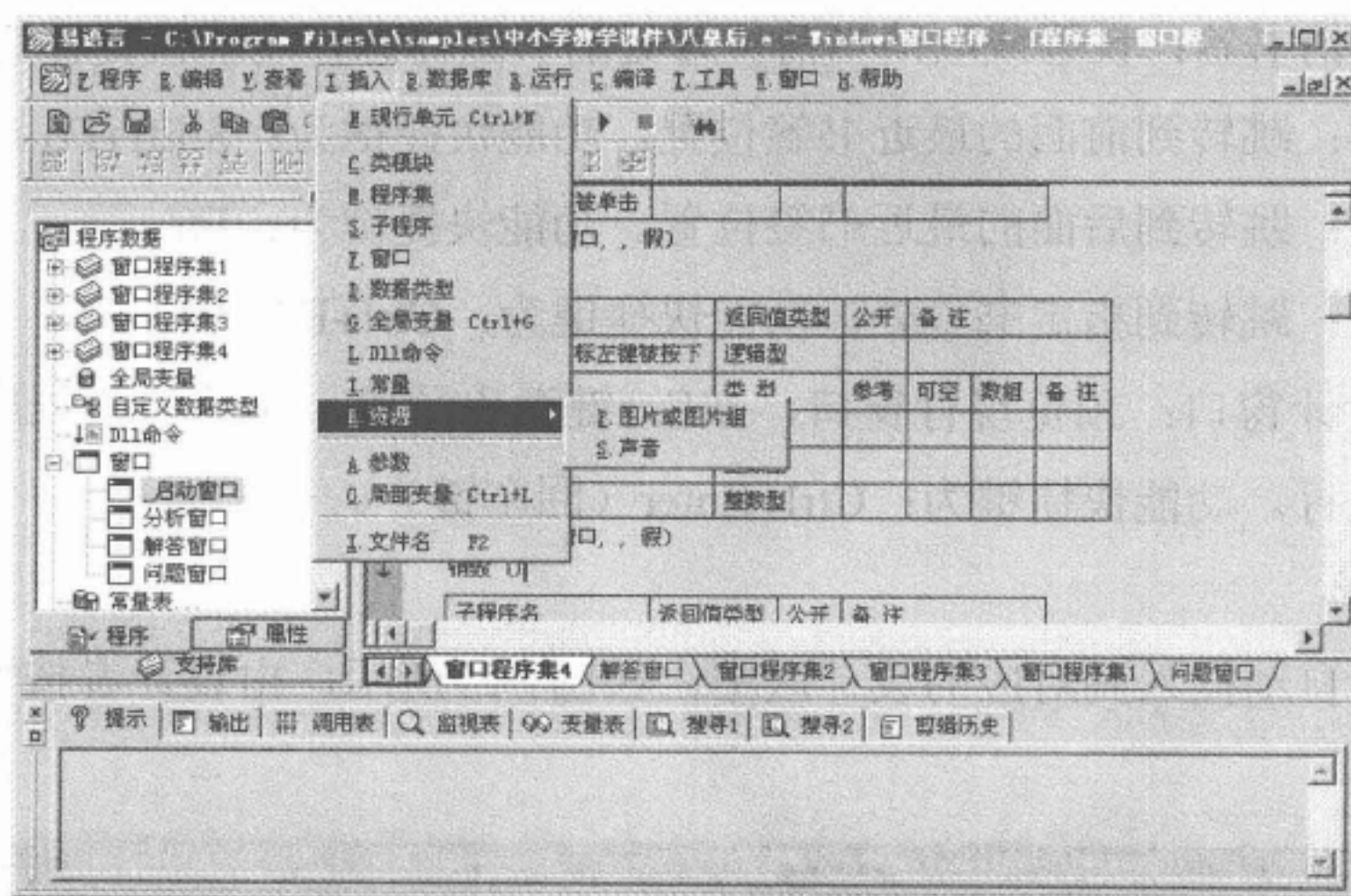


图1-28 资源子菜单

- 参数：插入一个新参数到程序中当前命令或子程序调用的参数表。
- 局部变量：插入一个新局部变量到子程序局部变量表。功能快捷键为：Ctrl+L。
- 文件名：在当前编辑光标位置处插入所选择文件的全路径名称。功能快捷键为：F2。

#### 5) 数据库菜单

此菜单下提供了对易语言数据库操作相关的工具，如图1-29所示。

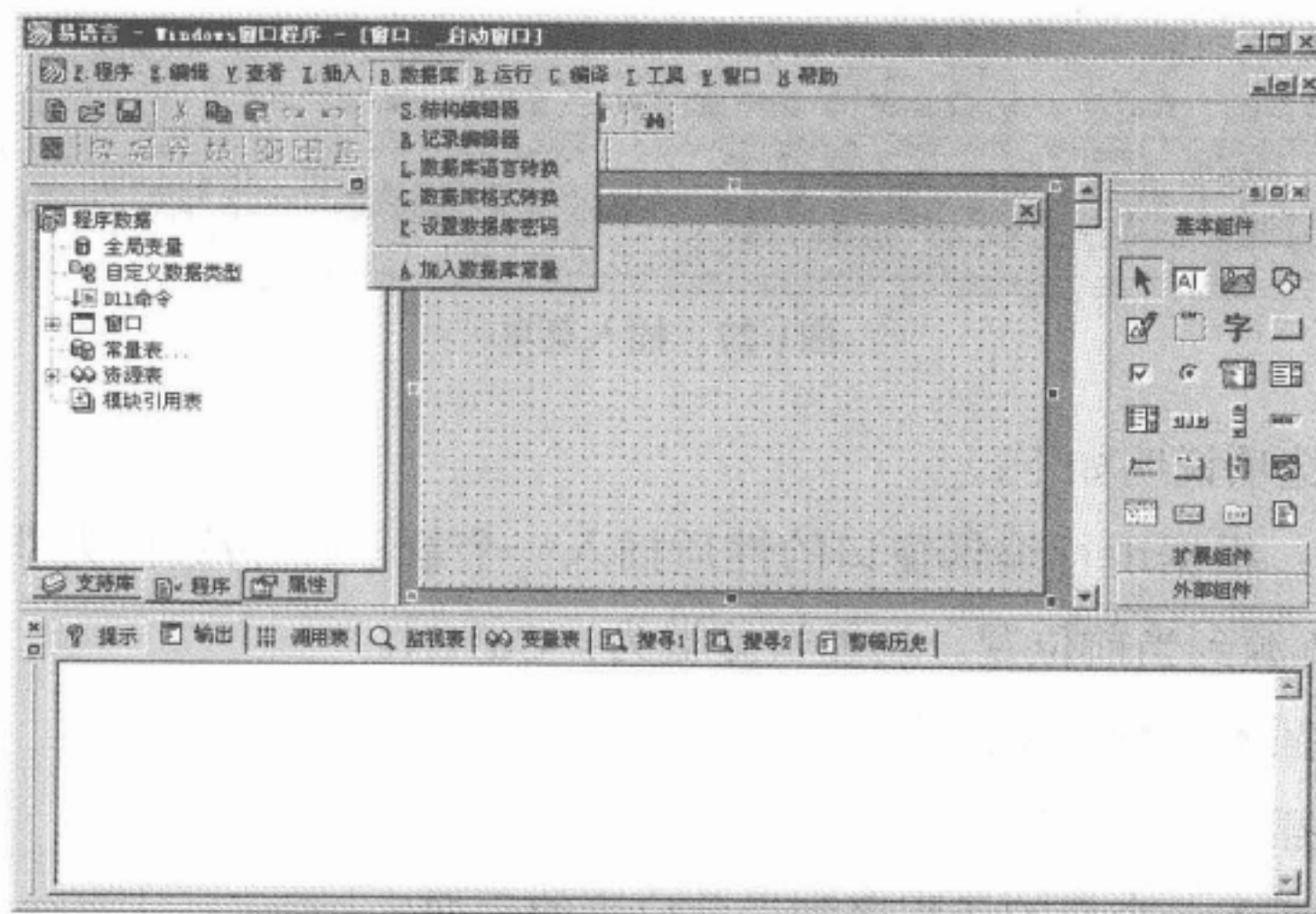


图1-29 数据库菜单

数据库菜单各选项说明如下：

- 结构编辑器：浏览或修改指定数据库的结构。本功能由数据库管理器.e（易语言安装目录\samples\基本例程\）编译后的程序提供，用户可以根据需要进行修改。
- 记录编辑器：浏览或修改指定数据库的记录。本功能由数据库管理器.e编译后的程序提供，用户可以根据需要进行修改。
- 数据库语言转换：将数据库中数据所使用的语言转换到另外一种。



- 数据库格式转换：可以将其他类型的数据库通过ODBC转换为易数据库。

ODBC（Open Database Connectivity）：开放数据库互连——是微软公司开放服务结构（WOSA, Windows Open Services Architecture）中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准（API）。

- 设置数据库密码：设置指定数据库的访问密码。

**注解：**数据库密码忘记、丢失将无法找回，也无法打开数据库，设置后请妥善保管。

- 加入数据库常量：将指定数据库的名称及所有字段名作为文本常量加入到系统常量表，以便在程序中使用。

## 6) 运行菜单

此菜单下提供了调试易语言程序的相关功能，如图1-30所示。具体使用方法将在本书后面章节详细讲解。

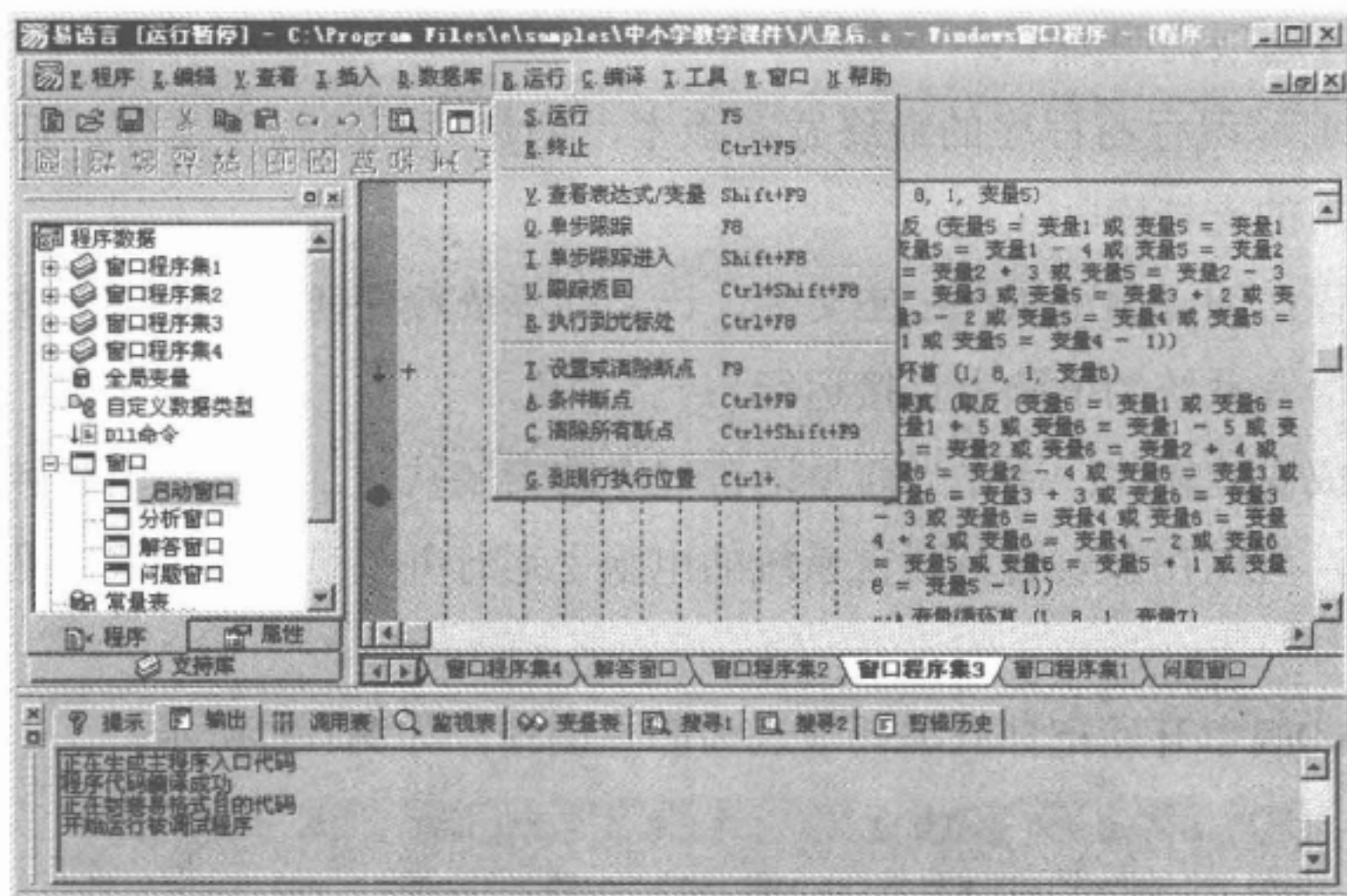


图1-30 运行菜单

运行菜单各选项说明如下：

- 运行：调试运行当前易程序。功能快捷键为：F5。
- 终止：终止现行调试运行的易程序。功能快捷键为：Ctrl+F5。
- 查看表达式/变量：查看/修改指定表达式或变量的内容。功能快捷键为：Shift+F9。
- 单步跟踪：在程序现行运行位置单步执行一行程序，如果此程序调用了子程序，系统不会跟踪到该子程序中去。功能快捷键为：F8。
- 单步跟踪进入：在程序现行运行位置单步执行一行程序，如果此程序行调用了子程序，则跟踪进入子程序。功能快捷键为：Shift+F8。
- 跟踪返回：跳出现行子程序（后面的代码会被执行），转到上级调用此现行子程序的语句后中断。功能快捷键为：Ctrl+ Shift+F8。
- 执行到光标处：运行易程序，在当前光标所在程序行处中断。功能快捷键为：Ctrl+F8。



- 设置或清除断点：设置或清除当前程序行处的断点。功能快捷键为：F9。
- 条件断点：设置或修改当前程序行处的条件断点。功能快捷键为：Ctrl+F9。
- 清除所有断点：清除掉程序中的所有断点。功能快捷键为：Ctrl+ Shift+F9。
- 到现行执行位置：跳到现行即将被执行语句的位置。功能快捷键为：Ctrl+.

## 7) 编译菜单

易语言在4.14版（包括4.14版）以前提供三种不同的方法，将源代码创建成.EXE、.DLL、.EC程序文件，如图1-31所示。



图1-31 编译菜单

编译菜单各选项说明如下：

- 编译：编译现行易程序的最终发布版本，创建对应的程序文件。但程序执行时需要所调用的支持库。
- 独立编译：编译出可执行EXE文件，该文件不依赖任何易语言系统文件，可以在未安装易语言系统的电脑上直接运行。
- 编译生成安装软件：制作当前易语言程序的安装软件，该软件不依赖任何易语言系统文件，可以在未安装易语言系统的电脑上运行并安装指定易语言程序，如图1-31所示。

易语言在5.0版本开始把独立编译改进为静态编译，如图1-32所示。

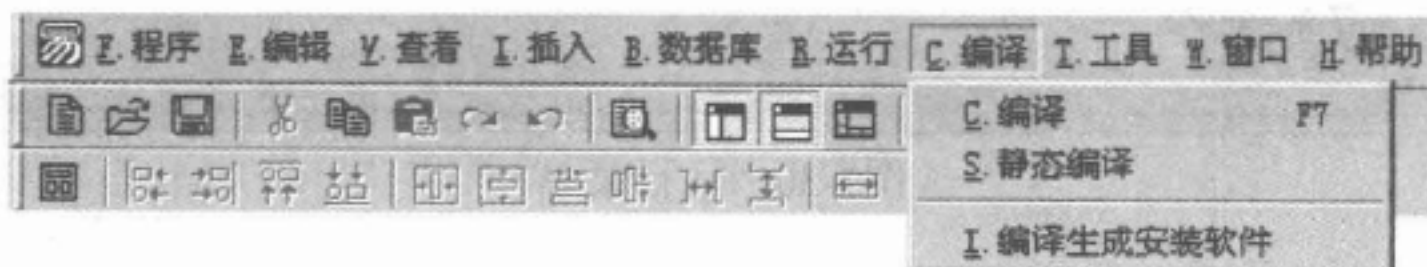


图1-32 编译菜单

## 8) 工具菜单

易语言系统提供的多种附加工具，如图1-33所示。可用来管理和配置易语言的扩展功能。

工具菜单各选项说明如下：

- 菜单编辑器：调用菜单编辑器编辑修改当前窗口的菜单。功能快捷键为：Ctrl+E。
- 报表编辑器：编辑报表模板文件。
- 执行易向导：执行指定的易向导文件。
- 矢量图形设计器：设计矢量图形供矢量图形设计器使用。
- 安装新的支持库：安装新的支持库或制作支持库安装包。
- 类型库或OCX组件->支持库：可以封装指定的COM（组件对象模型）库或OCX（对象链接和嵌入用户控件），使其能够在易语言中被使用。





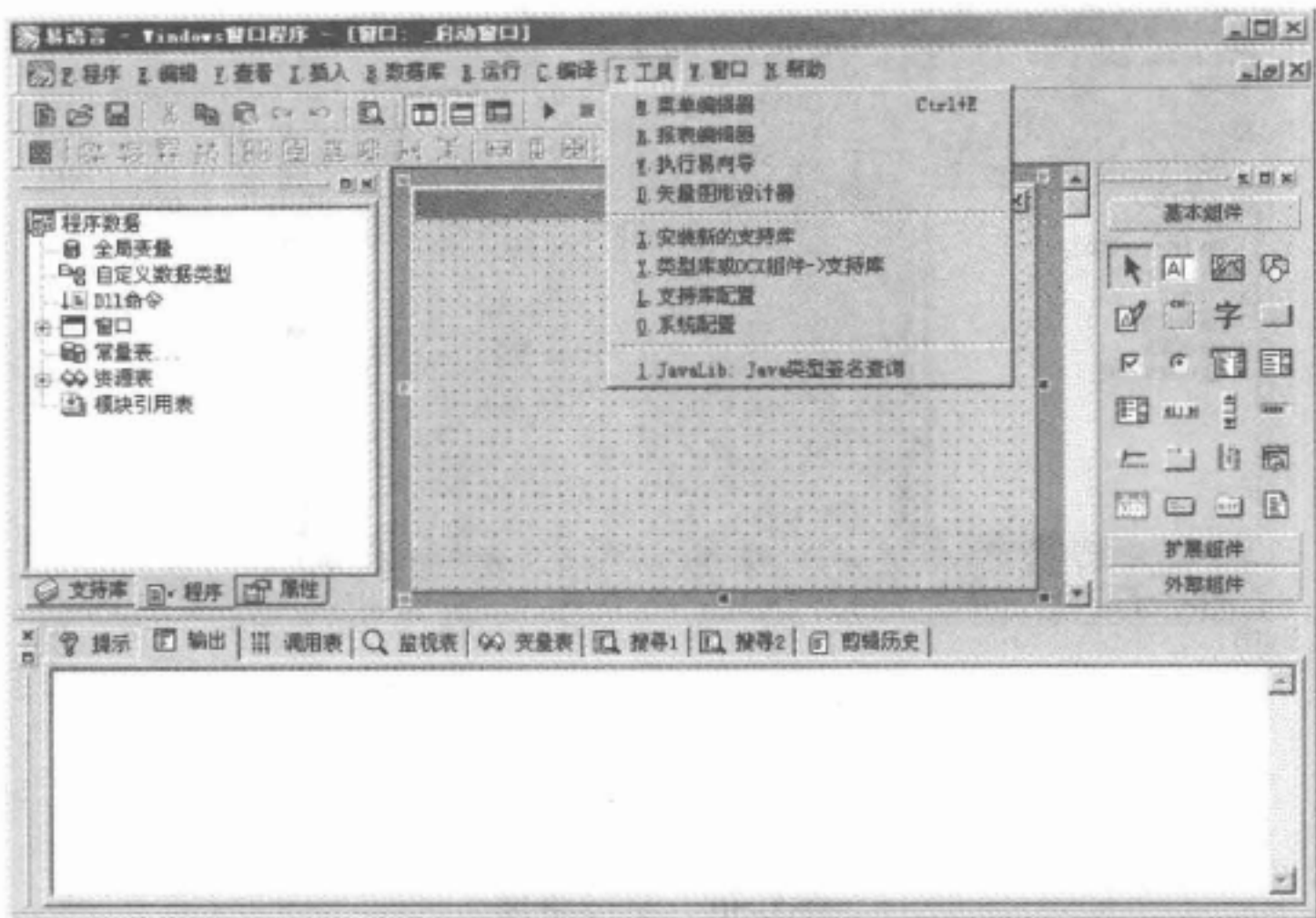


图1-33 工具菜单

- 支持库配置：易语言3.8以上版本初始安装后“支持库面板”默认只显示系统核心支持库，如图1-34所示。如果使用到更多的支持库需要在这里配置。  
点击支持库列表中任一项，可查看该支持库的基本信息。如果将某支持库名称前的对钩“√”加上，则在易语言的“支持库面板”列表中显示该支持库，并可以在程序中使用其提供的命令、数据类型、变量等。
- 系统配置：这里可以根据个人习惯，进行相关配置，如图1-35所示。通过调整该对话框中各项属性的参数，可以自定义界面各部位颜色，也可以选择各种配色方案，还可以更改代码字体和对内置输入习惯等很多方面进行配置。

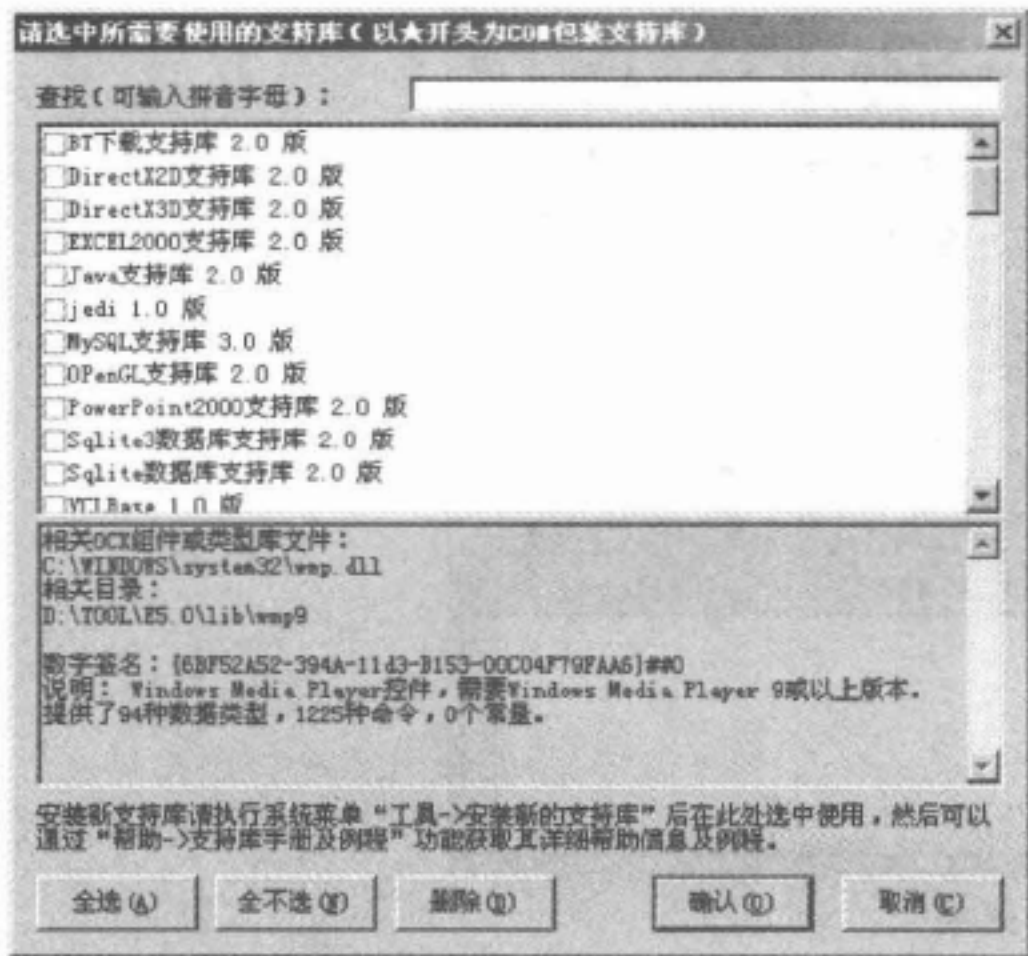


图1-34 支持库配置

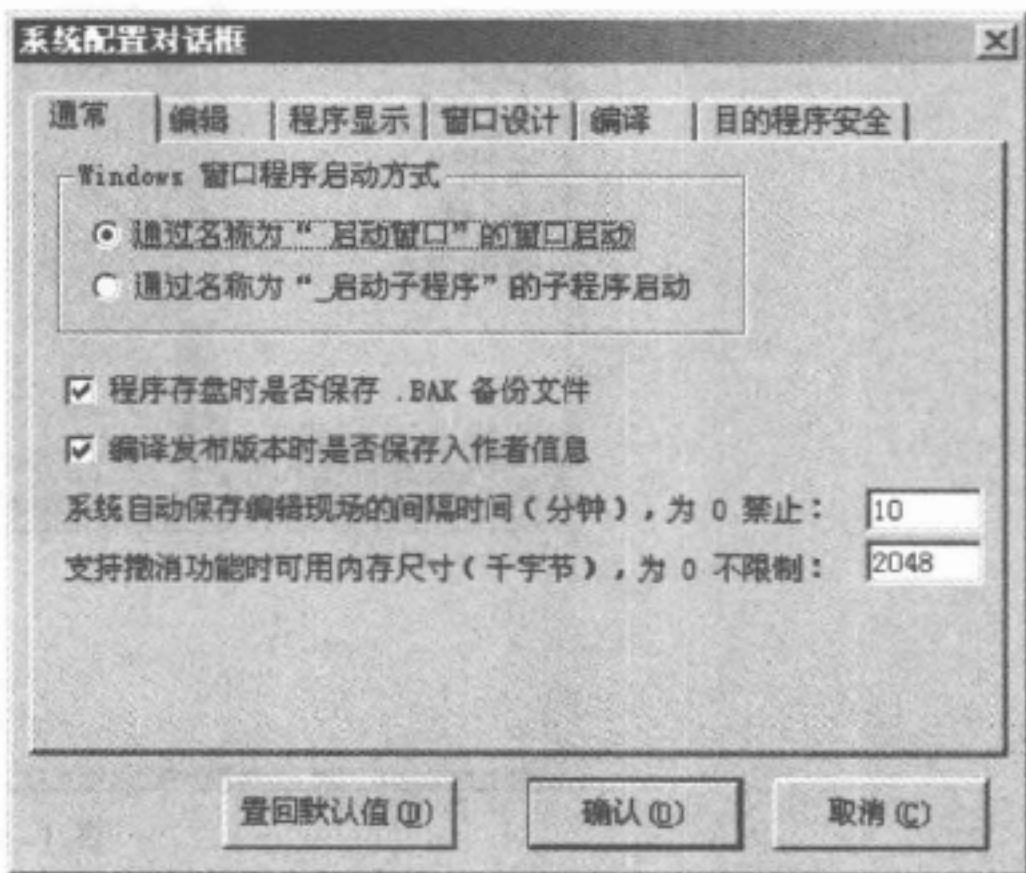


图1-35 系统配置对话框

## 9) 窗口菜单

设置已被载入设计窗口的排列方式以及已被载入设计窗口的名称、隶属和类型，如图1-36所示。



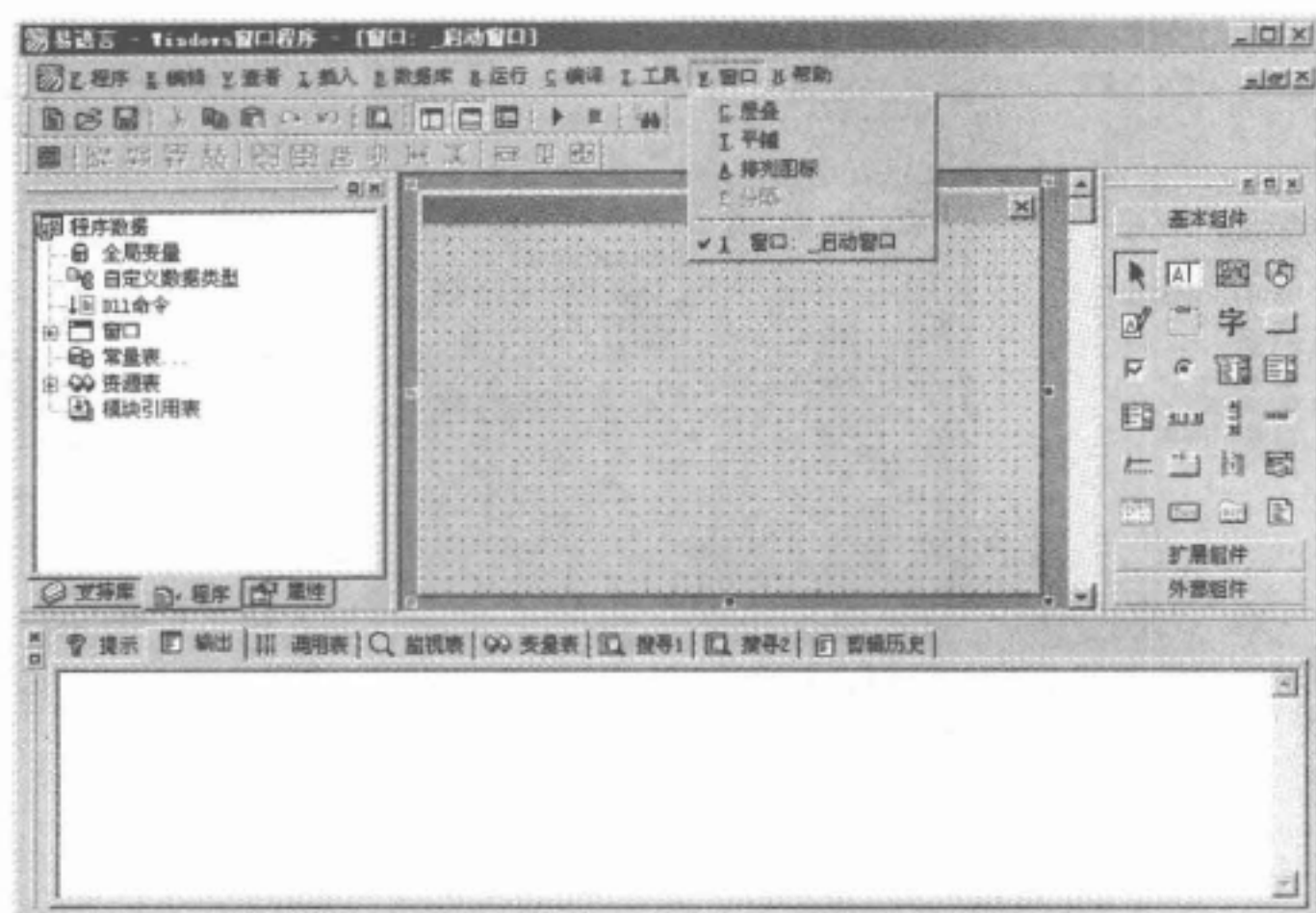


图1-36 窗口菜单

窗口菜单各选项说明如下：

- 层叠：排列窗口成相互重叠。
- 平铺：排列窗口成互不重叠。
- 排列图标：将图标排列在窗口底部。
- 分隔：将活动的窗口分隔成窗格。
- 窗口：已被激活的设计窗口。

#### 10) 帮助菜单

在窗口的“帮助”栏中有多种选项，如图1-37所示。

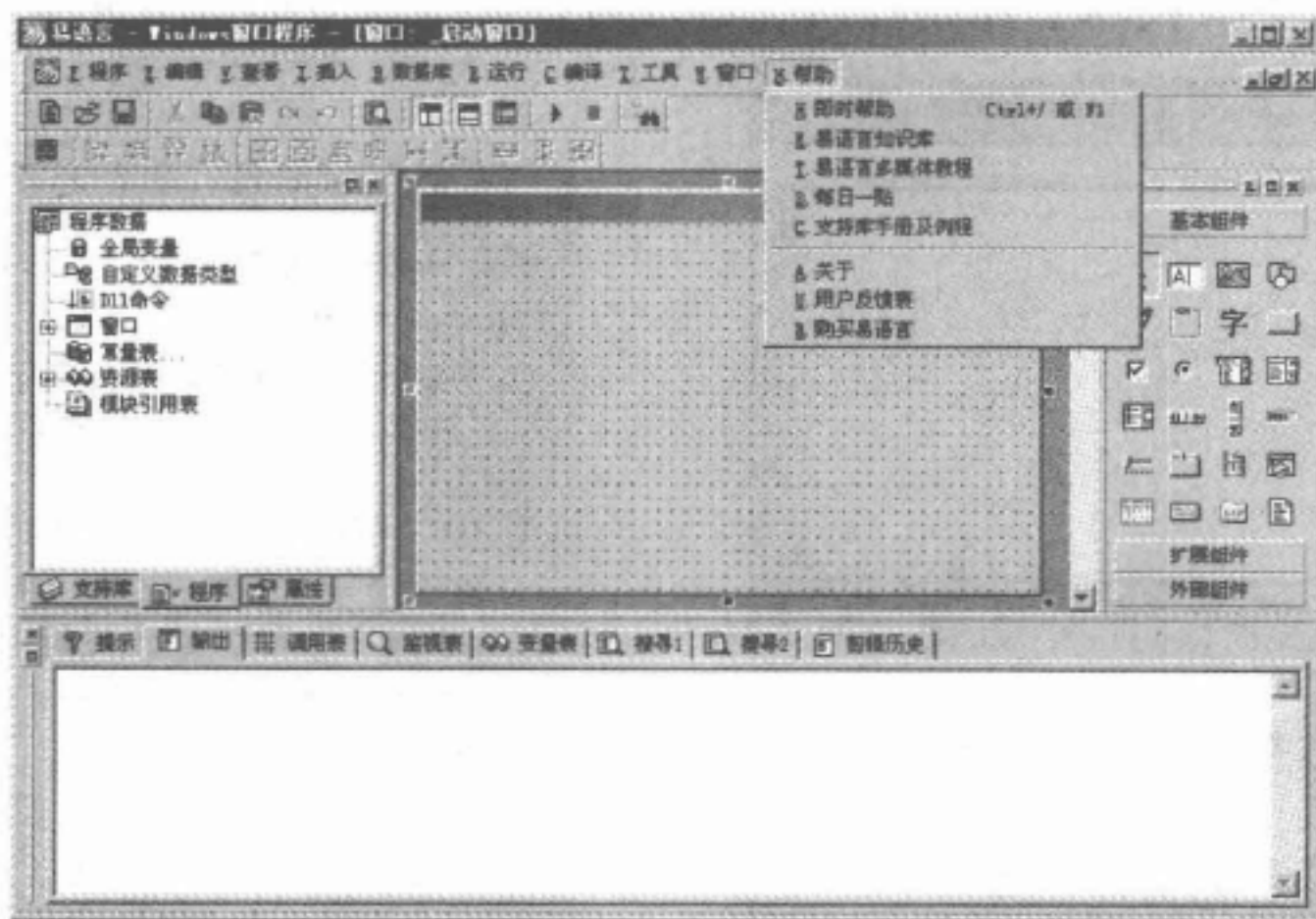


图1-37 帮助菜单

帮助菜单各选项说明如下：

- 即时帮助：在状态夹中显示有关当前位置的帮助信息。即时帮助信息内容是在用户进行任何操作的同时，随时按下“F1”热键将有关的支持信息在提示面板中显示出来。





- 易语言知识库：打开并显示易语言知识库。易语言知识库文档基本包含了当前易语言版本中所有帮助信息，并且配有大量的贴图和源代码，为学习易语言提供了很大帮助。

**注解：** 易语言知识库需要安装知识库文件，方可使用。

- 易语言多媒体教程：打开并显示易语言多媒体教程。
- 每日一贴：显示每日一贴。
- 支持库手册及例程：提供有关易语言支持库的帮助信息。
- 关于：显示当前易语言的版本号和版权信息。
- 用户反馈表：填写并递交用户反馈信息表。
- 购买易语言：购买易语言或查看是否已经购买成功。

## 1.4 代码编辑环境

通过对易语言整体菜单的介绍，读者对易语言已经有了初步的了解和认识。下面将讲解在代码编辑环境中需要提前了解和掌握的一些知识，通过对以下几点掌握，将有助于程序的开发和代码快速编写。

### 1.4.1 代码输入提示

新建一个“Windows窗口程序”，双击启动窗口，进入代码编辑区。在“\_启动窗口\_创建完毕”子程序下输入“rg”，在输入过程中，可以看到出现了一个如图1-38所示的是输入提示框，满足输入条件的命令就显示在输入提示框中。

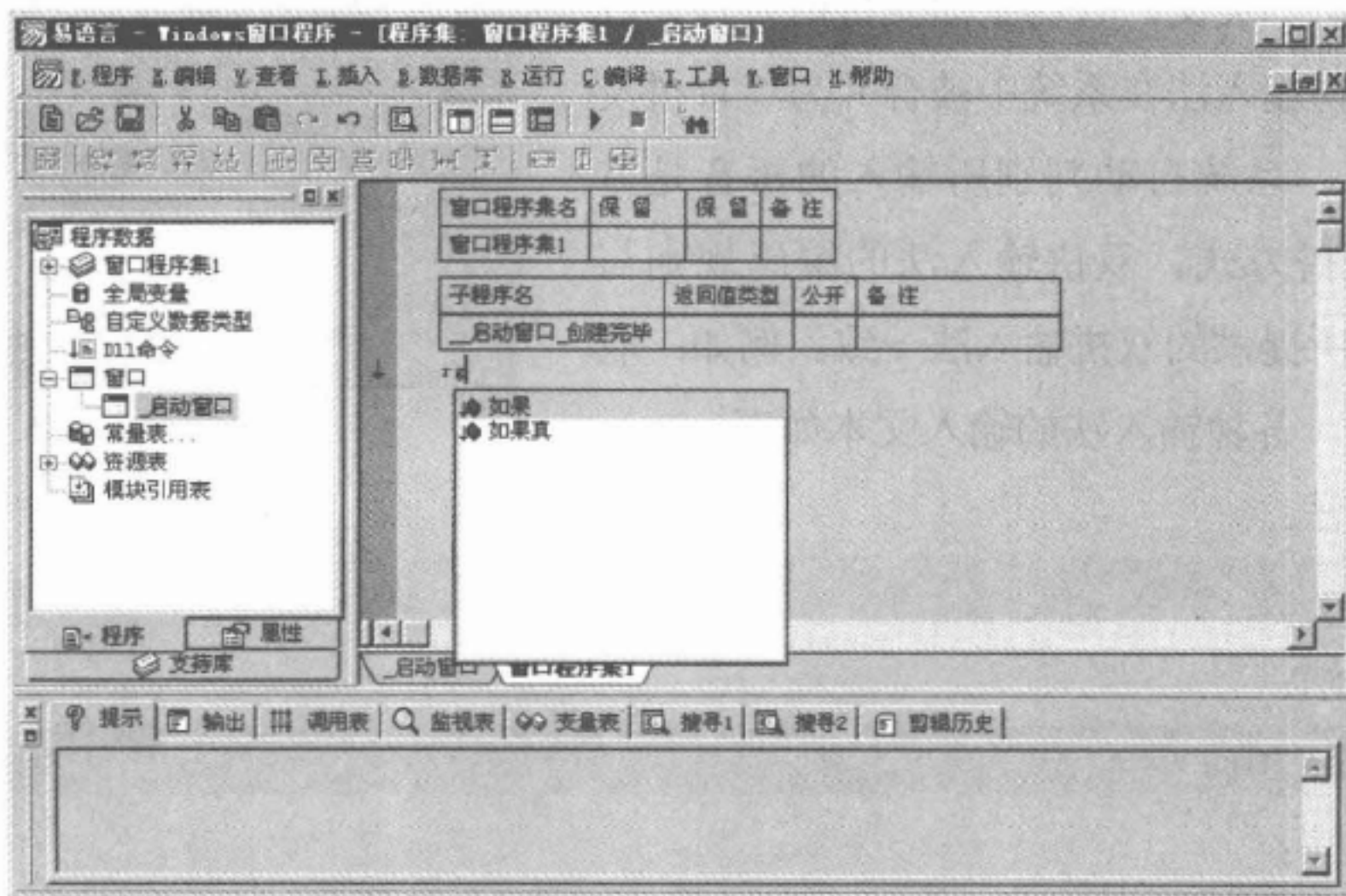


图1-38 代码输入提示



## 1.4.2 前层提示信息

易语言中，每输入一个命令代码，将鼠标移动到该命令上，都会出现一个信息提示框，如图1-39所示，显示该命令的帮助信息。

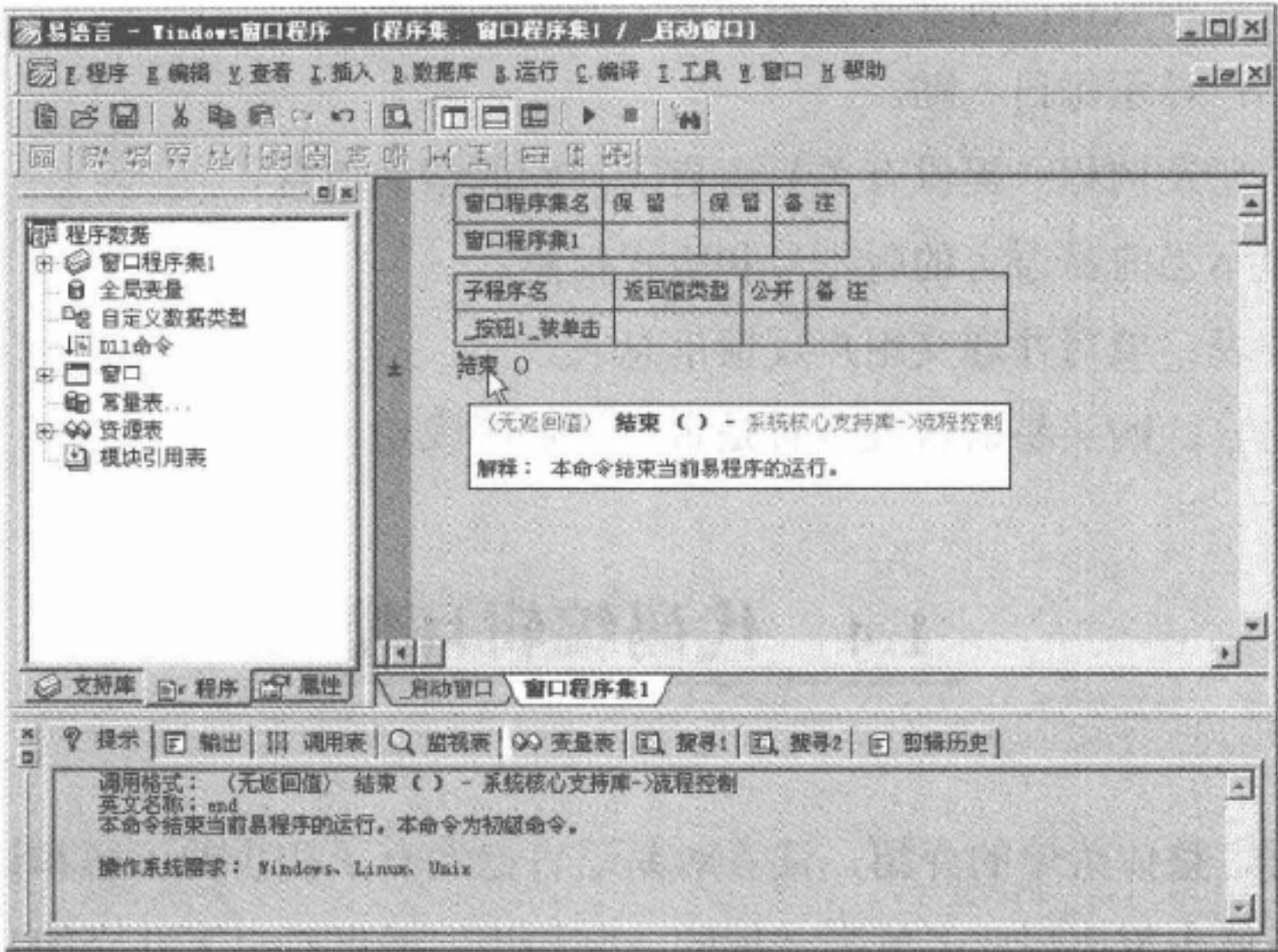


图1-39 易语言前层提示

## 1.4.3 代码输入方式

打开菜单栏“工具”→“系统配置”→“编辑”，如图1-40所示。

易语言内置四种习惯的代码输入方式：首拼、全拼、双拼、英文，且均全面支持南方音。首拼输入法及全拼输入法在系统中被合并为“首拼及全拼输入法”，系统自动判别所输入的拼音是首拼方式还是全拼方式。双拼输入法的编码规则与Windows系统所提供的双拼输入法一致。例如：取绝对值（20），各种输入法的输入文本如下：

首拼输入法：

qjdz(20)

全拼输入法：

qujueduizhi(20)

双拼输入法：

qujtdvvi(20)

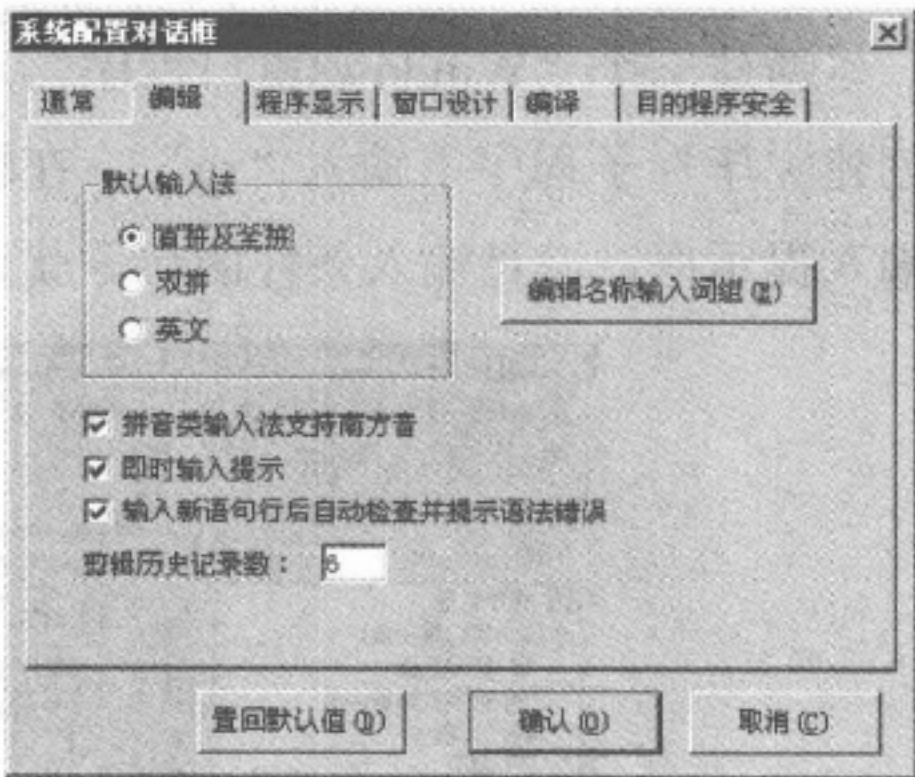


图1-40 代码输入方式



英文输入法:

abs (20)

代码中涉及的汉字, 都可以使用这几种内置输入法输入。

**注解:** 对于一些特殊情况, 对于没有声母的汉字 (如“按”), 使用首拼输入法时应取其韵母全部字母。比如: “按钮”, 用首拼输入法就可以输入成“ann” (其中“an”为“按”的韵母, “n”为“钮”的声母)。

如果要输入的名称中既有汉字又有英文字母, 则其中的英文字母不论大小写都要用大写英文字母输入。如: 要输入“编辑框a”, 使用首拼输入法就要输入: “bjkA”。

使用五笔字型、自然码、智能ABC等这些由Windows提供的系统中文输入法, 在易语言中也可以进行程序代码的输入, 只要在输入代码时打开该输入法即可 (直接输入中文代码)。

#### 1.4.4 参数分步输入

易语言提供的参数引导输入功能, 减少了记忆量, 节省编程的时间, 减少了程序录入的错误。对于参数较多的命令, 程序员不需要再花时间去查询参数的意义, 可以直接将命令展开输入, 方法如下。

将光标停在欲展开的命令行上, 如果当前行没有通过编译, 则不能展开命令, 可以使用Shift+Enter (回车键) 来预编译当前行, 然后按下ALT键+方向键的右键, 该命令就会被展开, 各参数都列在了该命令的下面, 可以直接在命令下的参数分支上输入, 如图1-41所示。

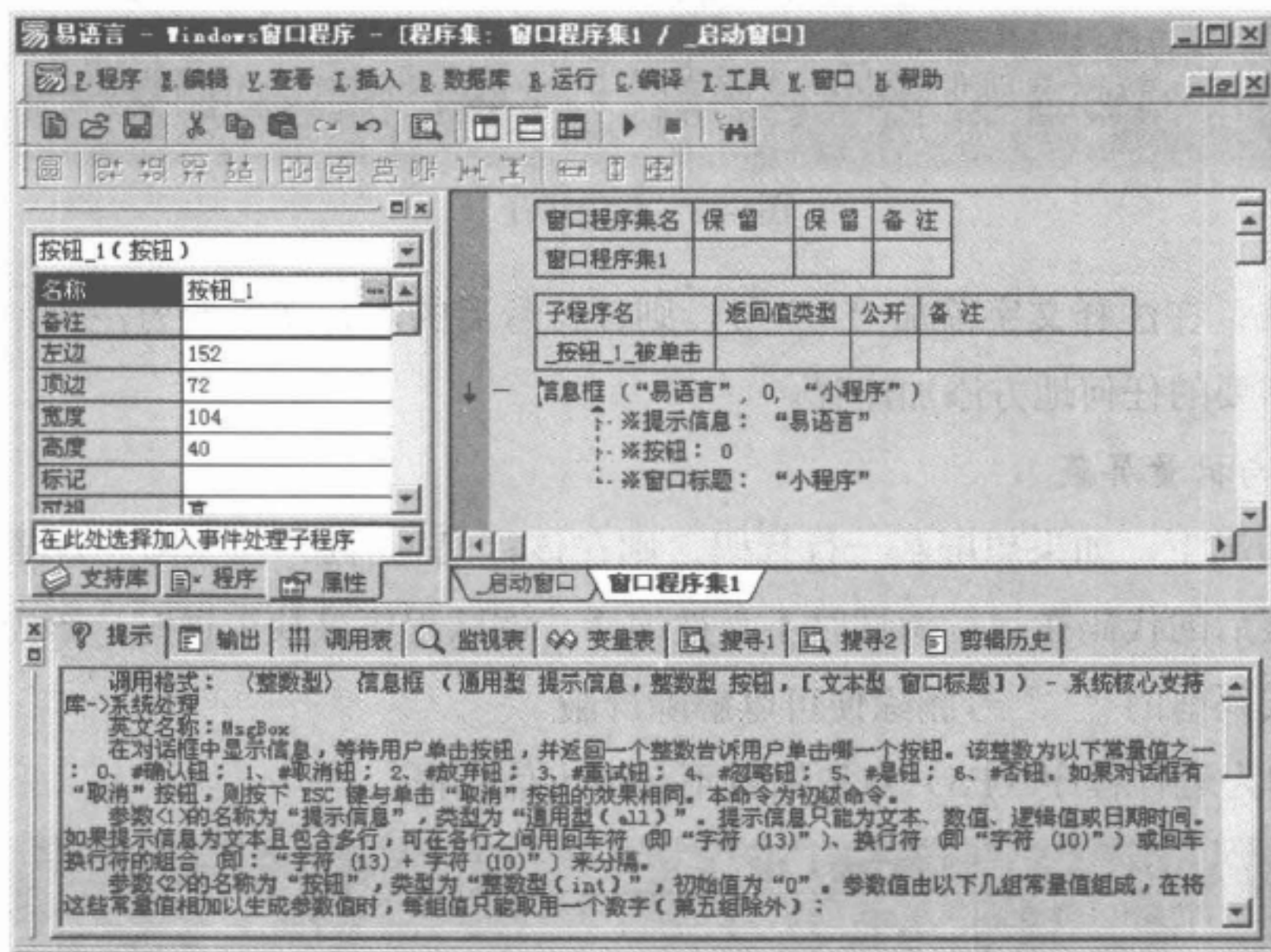


图1-41 分步输入参数

在状态夹-提示中将显示本参数的说明信息。



这里我们也可以运行（F5）一下如图1-42所示的小程序，看看显示的是什么呢？

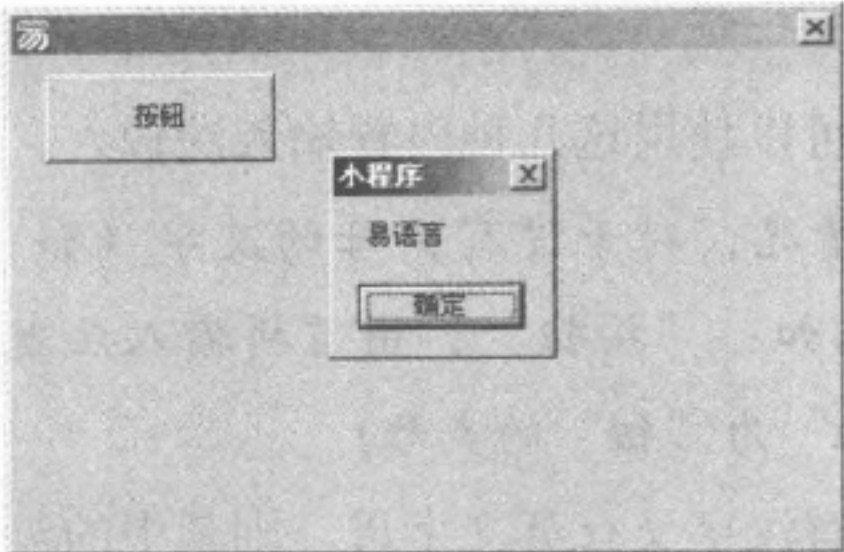


图1-42 运行显示

### 1.4.5 输入注释与代码屏蔽

#### 1) 注释输入

注释是一行或一段代码的提示和说明。编写代码时一定要养成一个良好的习惯，就是给部分代码输入注释信息，如图1-43所示。这样一来，既方便了自己日后阅读代码，又方便其他人更快地理解程序代码的思路和功能。

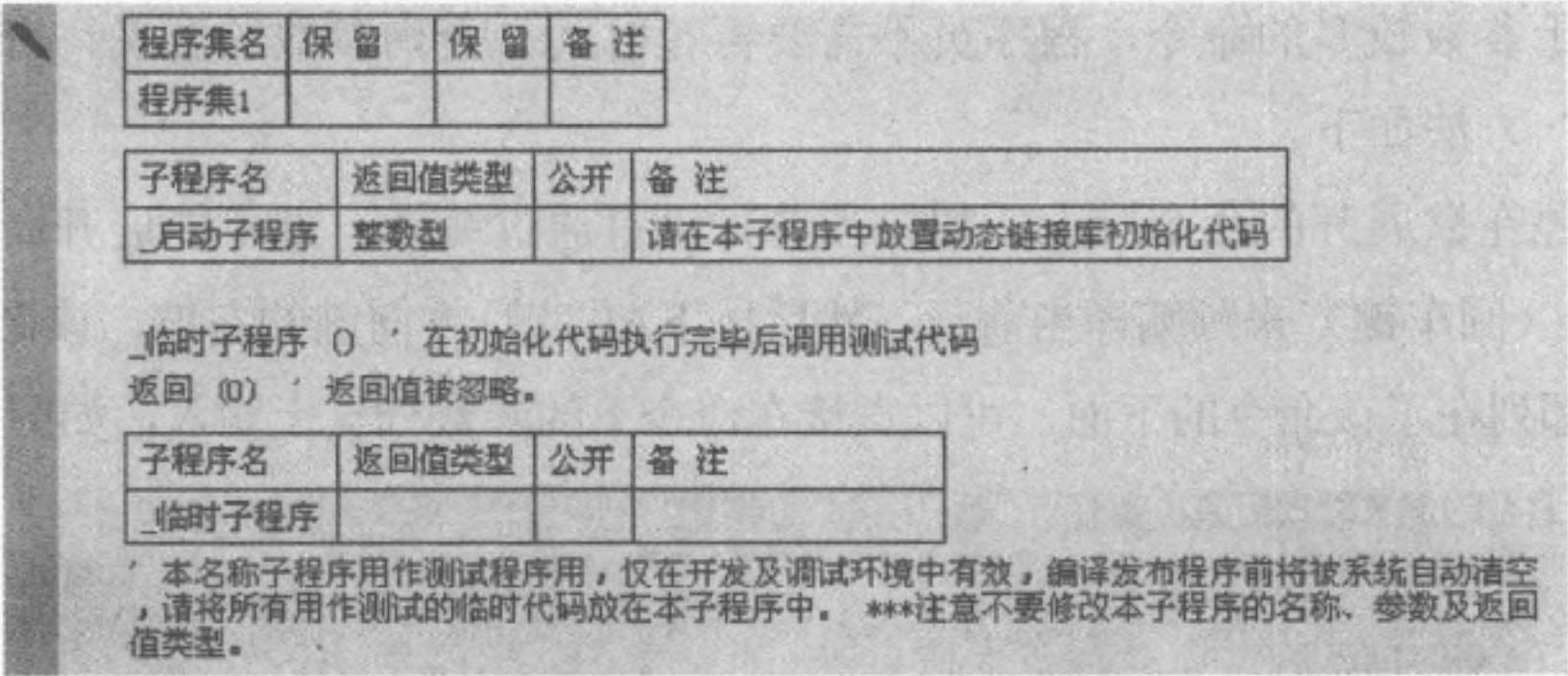


图1-43 代码注释

输入方法：在注释文字前加“'”号，则该符号后的本行文字变为注释，在输入代码时，可以在需要的任何地方添加注释。

#### 2) 屏蔽与批量屏蔽

在任何情况下，如果想屏蔽一行代码，则在该行代码前加“'”号，和置为注释的方法相同，屏蔽后的代码在运行调试时不会被编译，调试程序寻找错误时，该方法会起到很大作用。将代码前的“'”号删除便可以解除屏蔽。

如图1-44为运行程序（F5）点按钮就不会弹出信息框。

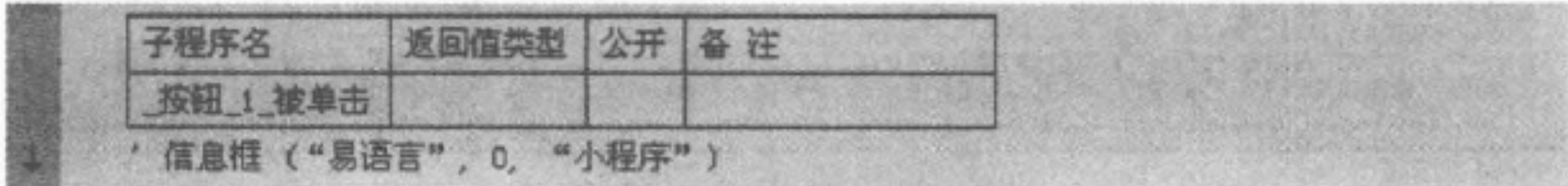


图1-44 屏蔽代码

还可以在代码上点击鼠标右键，弹出的菜单中也有“屏蔽”和“解除屏蔽”选项，快



快捷键是Ctrl+K键；选中多行代码然后按Ctrl+K键可以屏蔽，按Ctrl+M键可以解除屏蔽。

### 1.4.6 易语言语句分类

易语言常见的程序语句分为三种：赋值型语句、非运行语句、命令语句。

#### 1) 赋值型语句

即给某组件属性或某变量赋值的语句。一般使用“=”连接被赋值方和赋予的值，并且赋予的值一定要和被赋值的属性或变量的数据类型相同或互相兼容。

#### 【例】

##### (1) 给组件属性赋值：

标签1.标题=“易语言”

##### (2) 给变量赋值：

整数变量=100

#### 2) 非运行语句

即在运行过程中不被运行的语句，如注释型语句。

#### 3) 命令语句

即执行命令使用的语句。命令是一种程序运行动作指令。命令的调用格式为：[返回值][所属对象.]命令名称（[参数1]，[参数2]，...）。

中括号代表可以省略的部分。

#### 【例】

##### (1)

取文本左边（“123”，1）

##### (2)

编辑框1.加入文本（“内容”）

其中返回值是由命令执行后所决定回执给调用处的数据；所属对象表示该命令是由哪个对象提供的；命令名称是调用命令所必须的，用于在程序中标识命令；括号中的内容是命令的参数，主要是供命令进行判断、选择或再加工的因素，每个参数用逗号分隔。

### 1.4.7 易语言中的关键字

易语言中所有的命令名、组件的属性名都可被看做是易语言的关键字。在易语言中的组件名称、变量名称和子程序名称等都是可以自定义的，所以在起名称的时候既要清楚明确又要防止和这些关键字重名。虽然遇到有重名的系统会自动提示，但为了减少不必要的麻烦，还是要尽量避免重名。

### 1.4.8 书签

菜单栏“查看”→“设置或取消书签”“到前书签”“到后书签”“书签跳转”





在易语言中书签可以理解为标记，用于在代码中快速跳转到特定的位置。可用书签标记代码编辑区任何行。总之，书签在定位代码位置时会带来很多方便。

### 1) 设置或取消书签

要使用书签，首先在代码编辑区中插入书签，方法如下。

单击要插入书签的位置，选择 菜单栏 “查看” → “设置或取消书签” 或 快捷键 Ctrl+F6，如书签设置成功，光标所在行前将出现一个书签图标。

取消书签：在设置书签的位置重复设置一次书签即可取消已经设置好的书签，如图 1-45 所示。

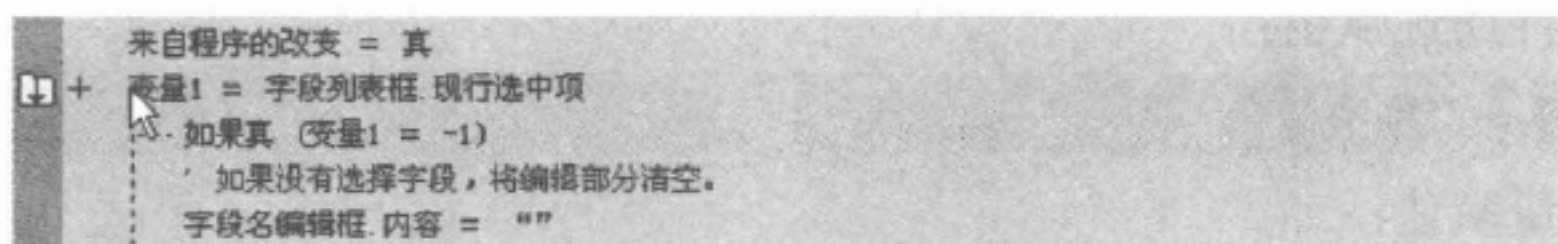


图1-45 设置书签

### 2) 到前书签

跳转到光标所在位置前面最近的书签位置。快捷键 Shift+F6。

### 3) 到后书签

跳转到光标所在位置后面最近的书签位置。快捷键 F6。

### 4) 书签跳转

跳转到已设置的所有书签中的指定书签位置。菜单栏 “查看” → “书签跳转” 或 快捷键 Ctrl+F7，在弹出的“书签跳转”对话框中，如图 1-46 所示，鼠标双击需要跳转到的位置即可。

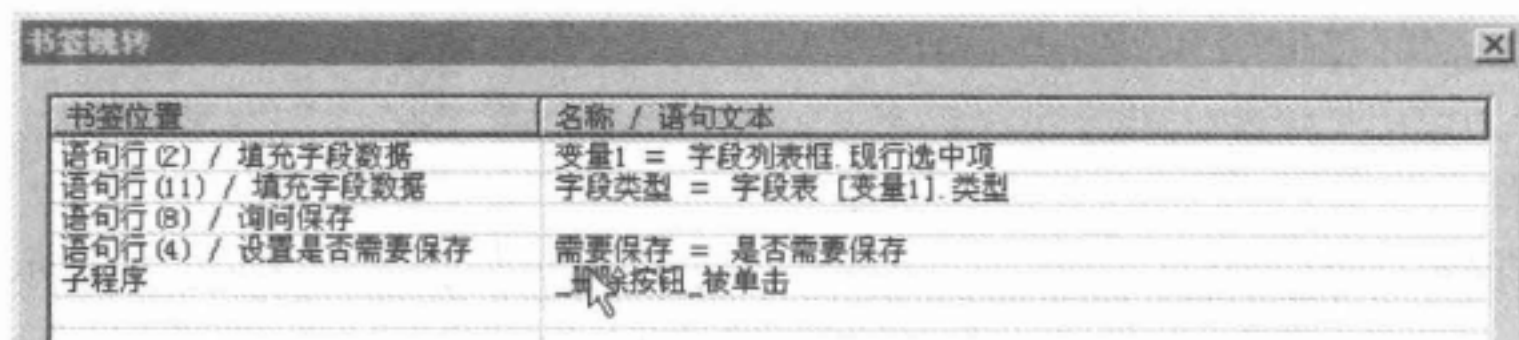


图1-46 书签跳转

## 1.4.9 即时帮助和帮助文档

易语言提供了即时帮助功能和内容丰富的帮助文档。用户根本不需要将全部的命令语法和参数含义都背下来。在实际开发工作中，可以使用命令分步输入方法（将语句展开），根据参数提示输入，或按 F1 键查看易语言的即时帮助，或查阅帮助文档。

在易语言的帮助系统中，所有的命令都有明确的分类。例如，想对一段文本进行操作，就要查找“文本操作”分类中的相关命令；如果想对一个文件进行操作，就要查找“文件读写”分类中的相关命令。

在易语言的支持库面板中，双击展开任意一个支持库名称，可以查到该支持库的所有





命令分类：双击展开其中任意一个分类名称，可以看到属于该分类的所有命令；点击任意一个命令名称，就可以在状态夹的提示面板中看到该命令的即时帮助信息，如图1-47所示。

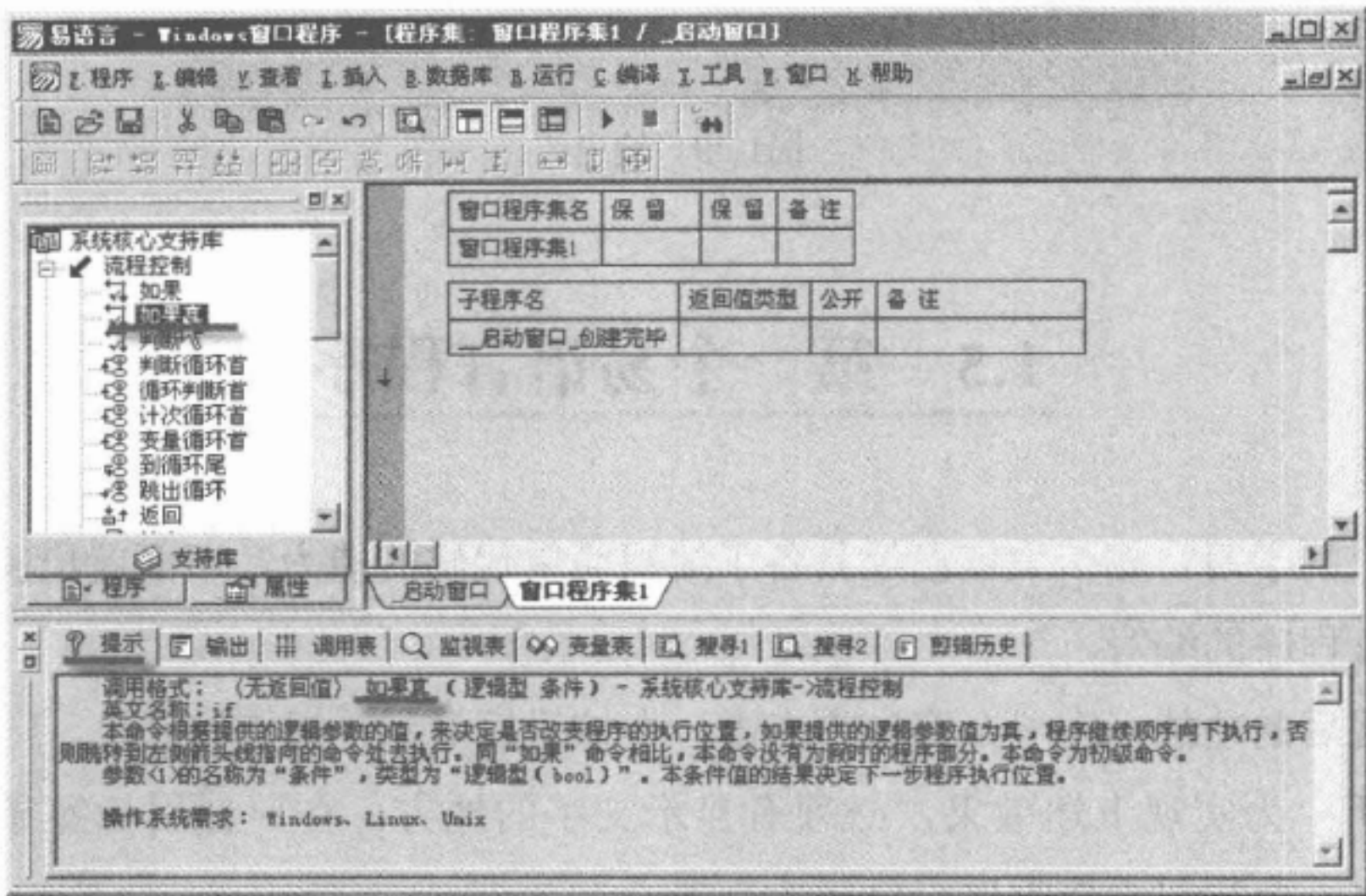


图1-47 即时帮助和帮助文档

在编写程序时，如果忘记了某个命令参数的含义，可以将光标停在该命令上，然后单击F1键，也可以在提示面板中看到该命令的即时帮助信息。

点击菜单“帮助”→“易语言知识库”，可以查看完整的帮助文档。其内容相对即时帮助更加丰富，并且每个命令都有简单的例程，查找命令也非常方便，如图1-48所示。

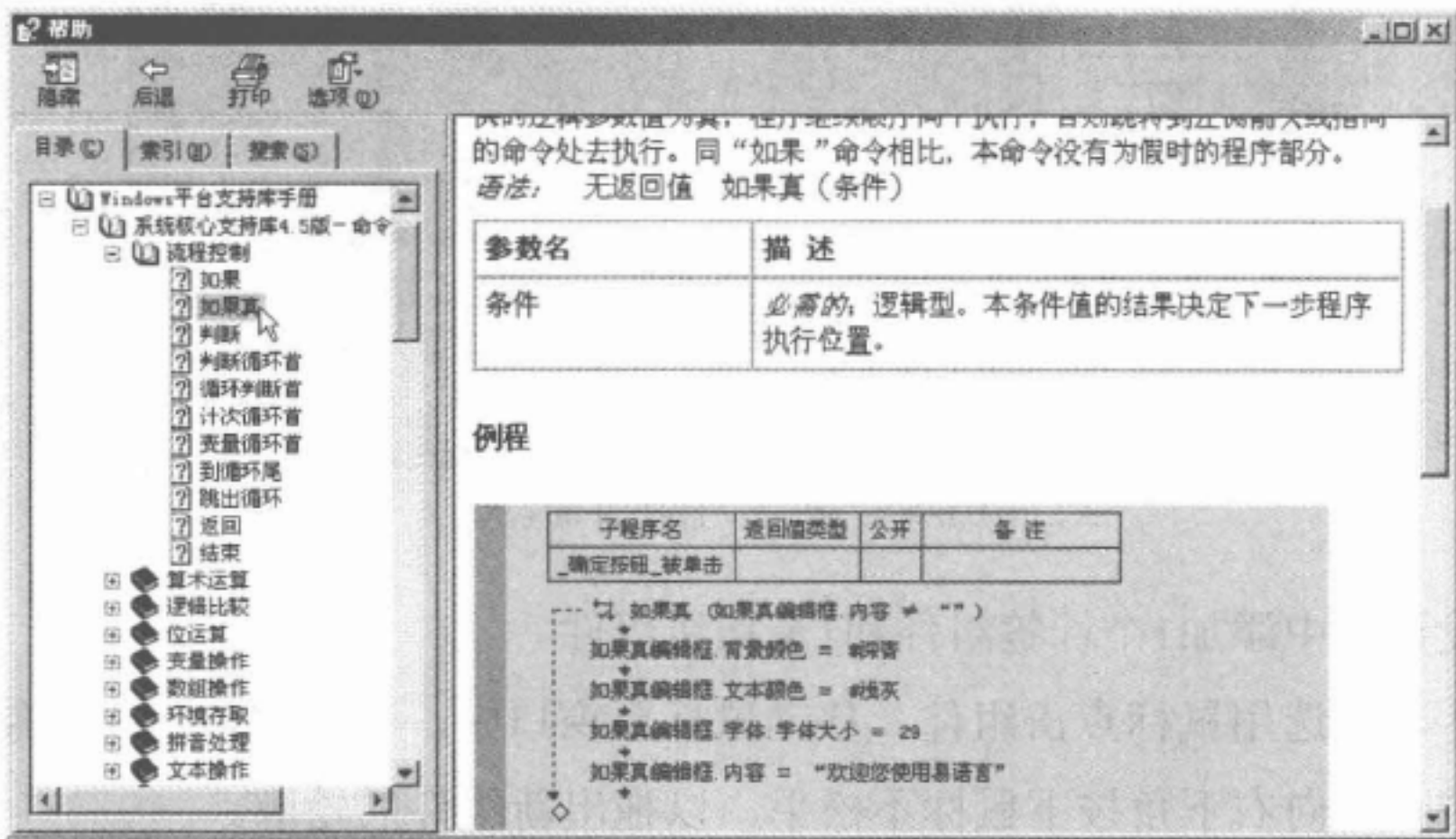


图1-48 易语言知识库

易语言核心支持库中已经提供了六百多个基本命令，各扩展支持库中也包含了很多命令。在平时的学习和开发中要注意活学活用，不要死记硬背，多使用分步输入法和即时帮助这两个功能来帮助记忆。

**注解：**当输入完一行代码时，使用组合键Shift+Enter即可检查目前编辑的这行代码有无语法错误。检查的结果将在“输出”栏显示，如图1-49所示。



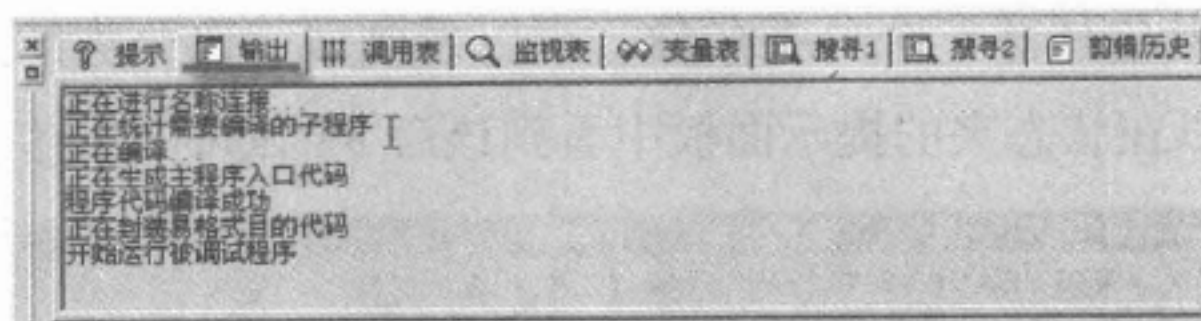


图1-49 输出

## 1.5 第一个易语言程序

通过以上的学习，相信大家已经具备了编写简单代码的能力，下面就检验一下，开始第一个易语言程序的编写。

**程序实现后的功能：**在一个窗口中点击一个按钮后显示“祖国您好”。

**功能分析：**为实现上述效果，必须有显示文字的地方。在此使用标签显示文字，当然，还需要有一个按钮来接收用户鼠标的点击。

**第一步：**新建一个易程序，即新建窗口中的“Windows窗口程序”，如图1-50所示。

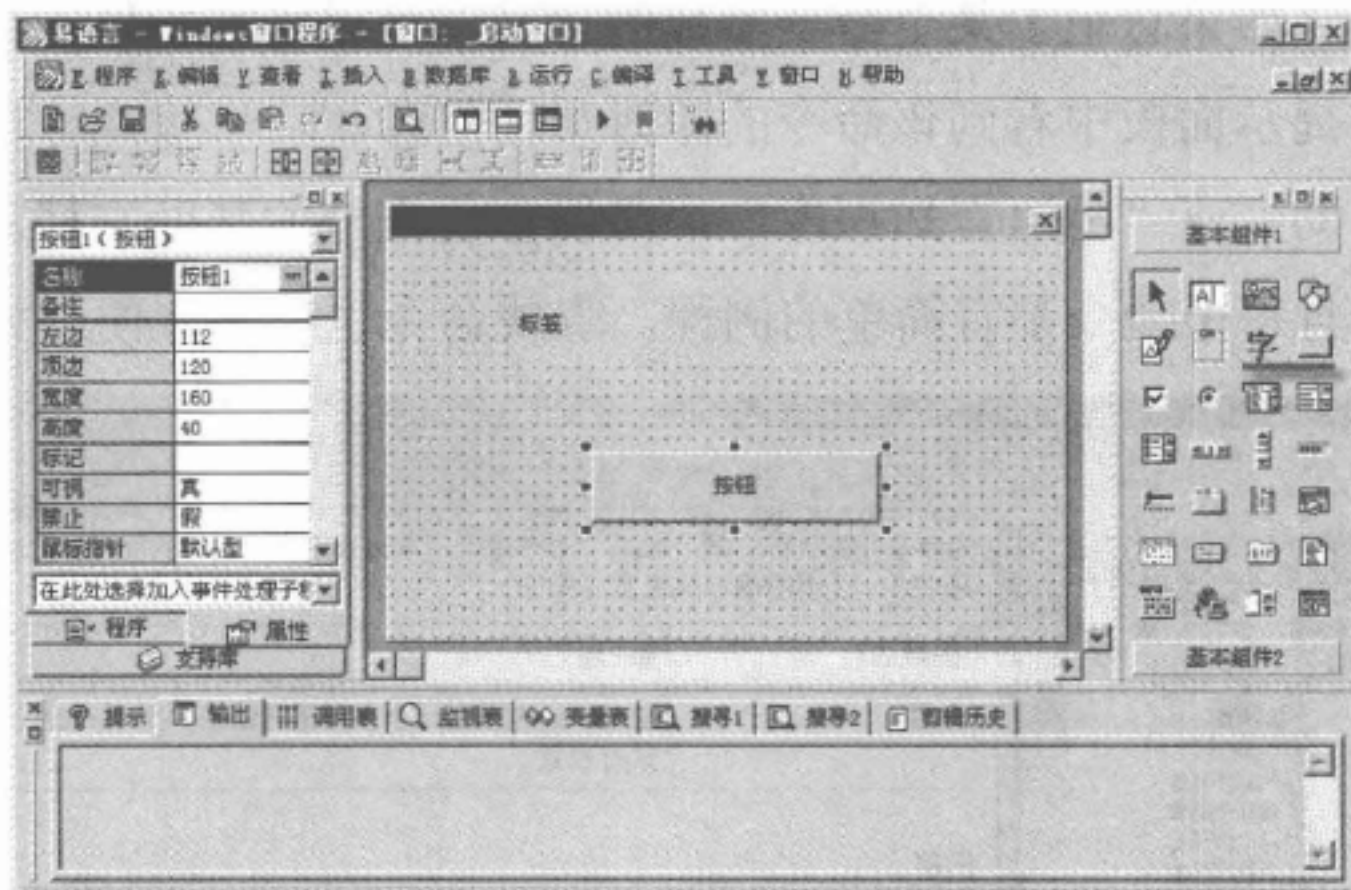


图1-50 第一个程序界面设计

在窗口设计区中添加1个标签组件和1个按钮组件。

添加组件时，选用鼠标点击组件，移动鼠标至窗口中，再在窗口空白处象画一个矩形框一样，从左上角向右下角按下鼠标不松手，以拖出新组件的轮廓。

**注解：**从组件箱中选出所需的组件添加到设计窗口中，只需要用鼠标左键在组件箱中点击欲添加的组件，使其处于选中状态，然后在设计窗口中按住鼠标左键拖动，即可完成该组件的添加。添加后的组件可以通过拖动鼠标改变其位置和大小，也可以使用方向键来调整组件的位置，还可以按住Shift键+方向键来微调组件的大小。

**第二步：**用鼠标双击窗口中的按钮组件，切换到代码编辑界面，并自动生成“\_按钮1\_被单击”子程序。然后在光标处输入以下代码。





标签1.标题 = “祖国您好”，如图1-51所示。

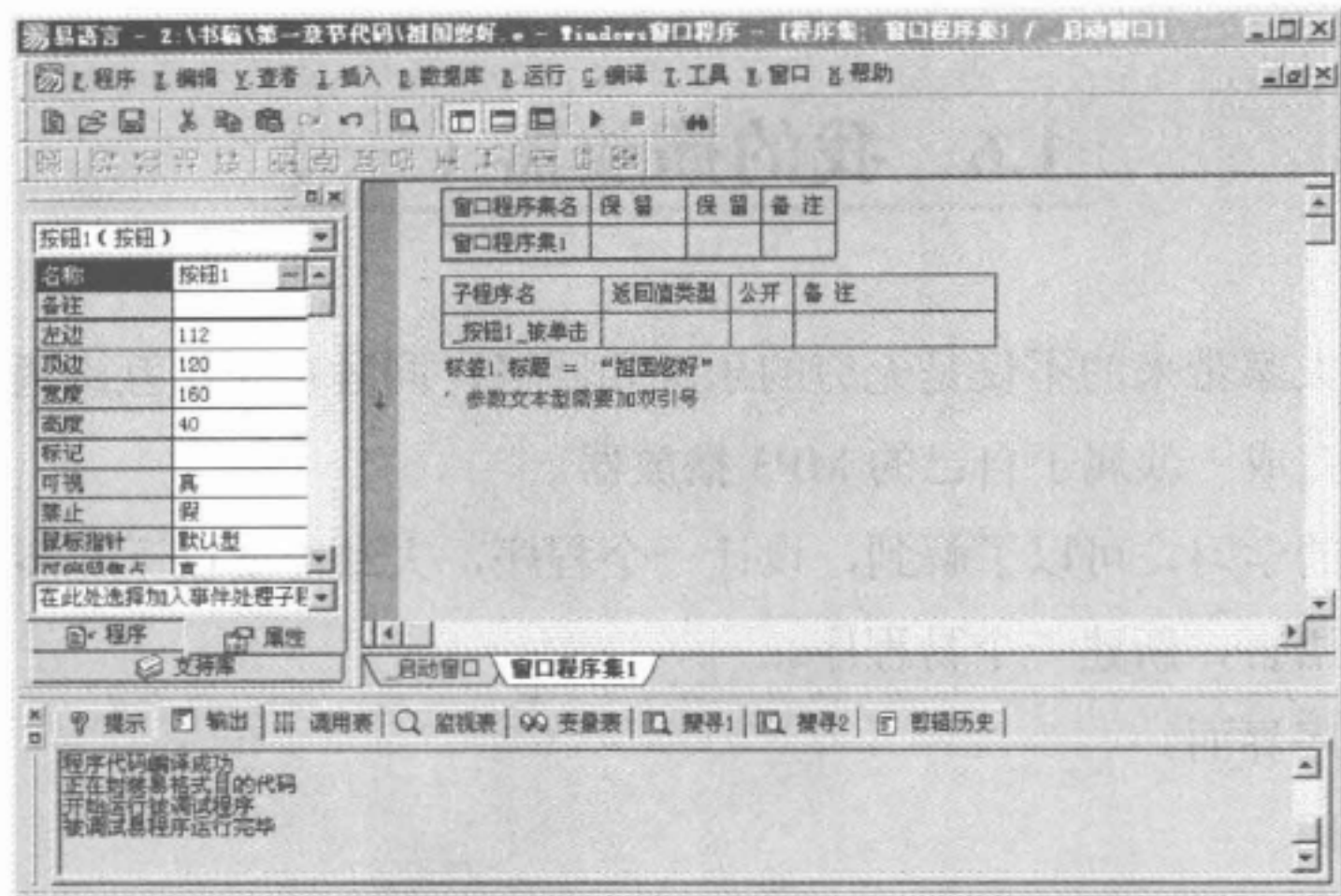
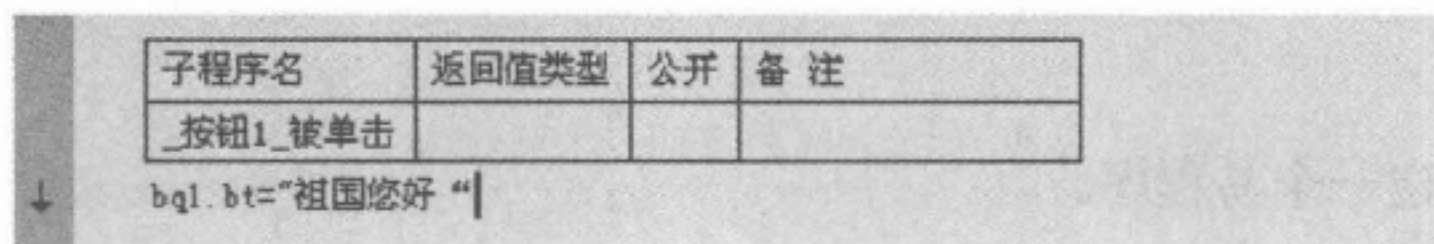
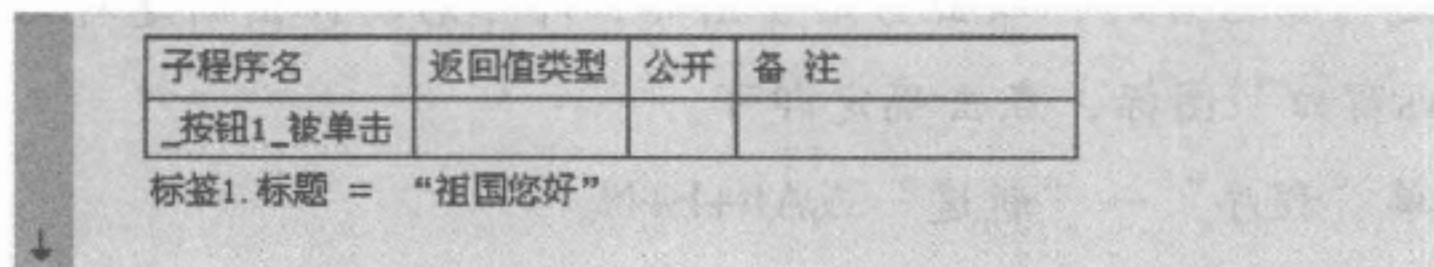


图1-51 在“\_按钮1\_被单击”子程序中输入代码

**注解：**输入代码也可以按照以下方式输入：



第三步：回车后将自动转换为：



第四步：代码输入完毕后，就可以试运行程序了。可以点击易语言工具条中的运行按钮，也可以按下F5键来运行程序。程序运行后点击窗口中的按钮，标签显示出如图1-52所示的“祖国您好”。

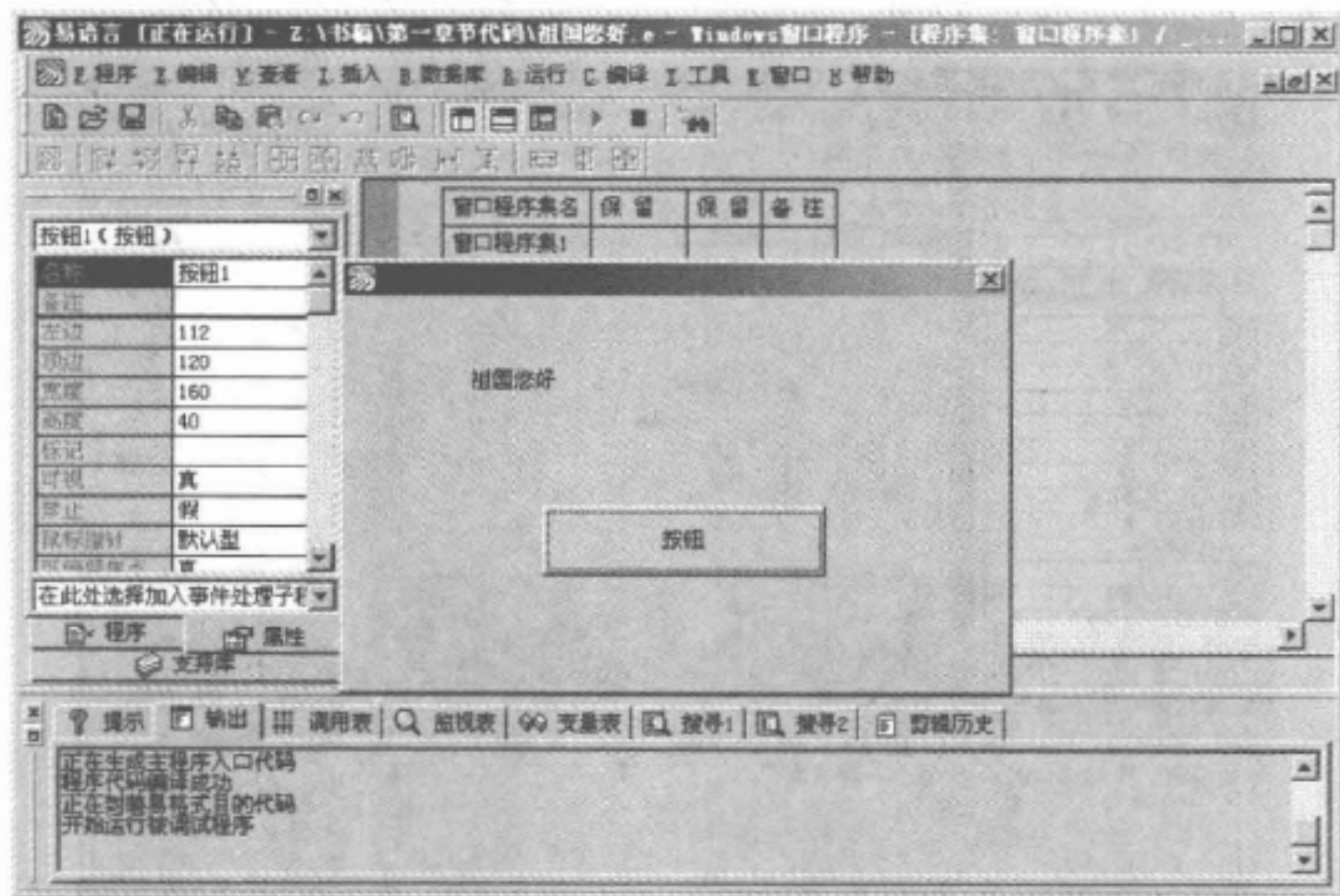


图1-52 运行结果



程序可以通过“编译”菜单编译为EXE文件发布。程序的发布会在后面的章节详细讲解。

## 1.6 我的播放器（一）

相信音乐给大家带来的不仅是无穷的乐趣和回忆，同时也是相互沟通的桥梁。跟随本书的学习我们将完成一款属于自己的MP3播放器。

通过本章的学习，可以了解到，设计一个程序，大致需要以下几步。

- (1) 启动易语言，新建一个易程序。
- (2) 设计程序界面。
- (3) 编写代码。
- (4) 调试、运行编写好的程序。
- (5) 生成可执行文件。（需要购买易语言商业版才可以进行编译）

这也是编写一个易语言程序的通用步骤，其中第2、3、4步可能重复多次，以修改与加强程序功能。

第一步：新建一个易程序。

**注解：**新建易程序可以有三种方法。

- (1) 在没有运行易语言时，双击易语言图标，即可启动弹出新建对话框，并在对话框中选中“Windows窗口”图标，点击确定即可。
- (2) 使用菜单“程序”→“新建”或Alt+F+N。
- (3) 使用工具栏“新建”按钮。

第二步：设计程序界面。

在新窗体中画一个按钮控件。

修改按钮属性中的名称为：按钮\_播放，按钮的标题为：播放按钮，如图1-53所示。

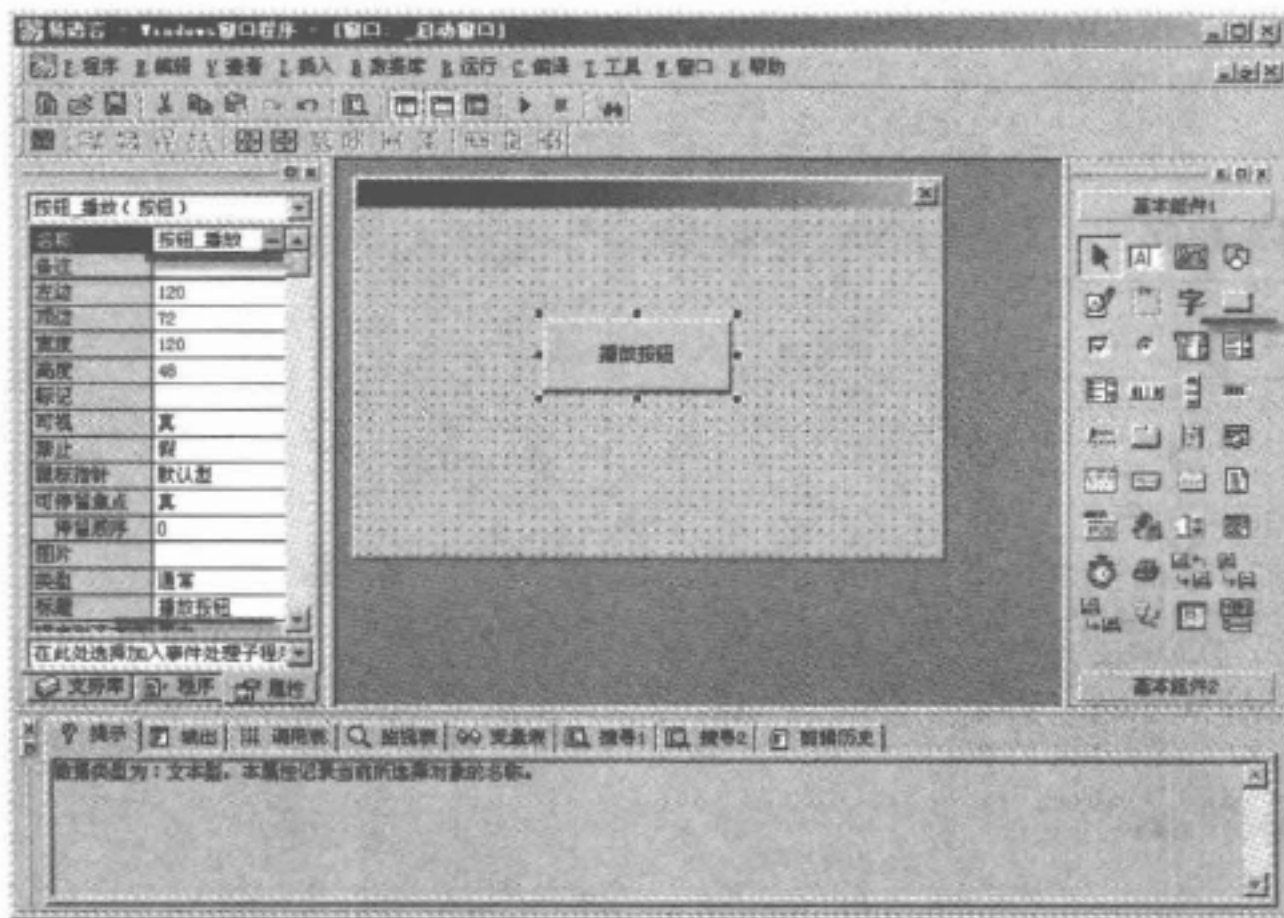


图1-53 MP3界面设计



第三步：编写代码。

鼠标双击窗口中的播放按钮组件进入代码编辑界面。然后在光标处输入以下代码。

播放MP3 (1, “在路上.mp3”)

**注解：**所播放的歌曲要和程序在同一目录下，如图1-54所示。

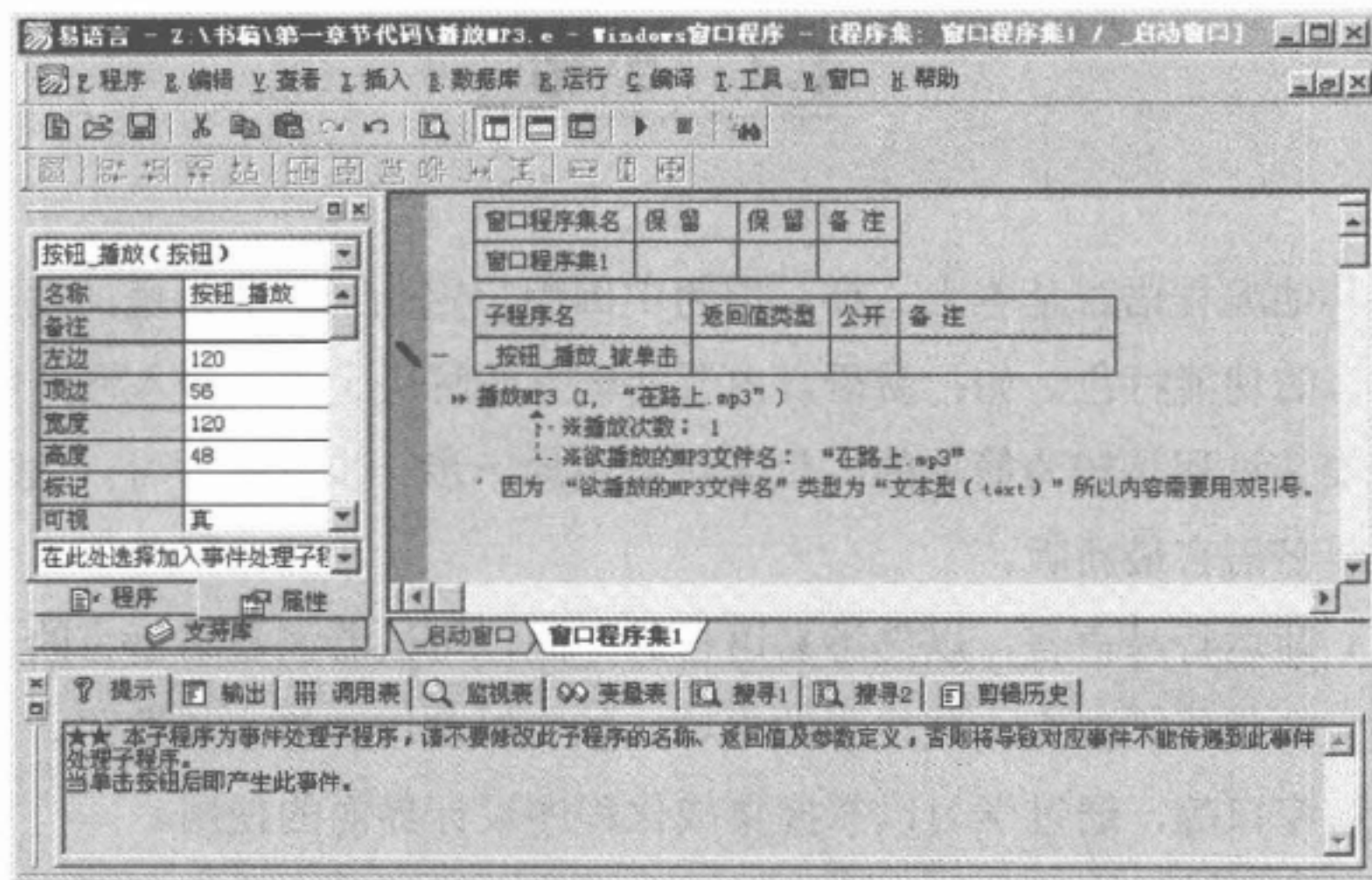


图1-54 在“\_按钮\_播放\_被单击”子程序中输入代码

思考：大家可以想一想不在同一目录下要如何处理？

第四步：运行编好的程序。

**注解：**有三种方法可以实现运行例程。

- (1) 菜单“运行”→“运行”。
- (2) 单击工具栏上的“运行”按钮。
- (3) 按热键F5。

运行程序 → 单击“播放按钮”，如图1-55所示。（不要忘记打开音箱。）

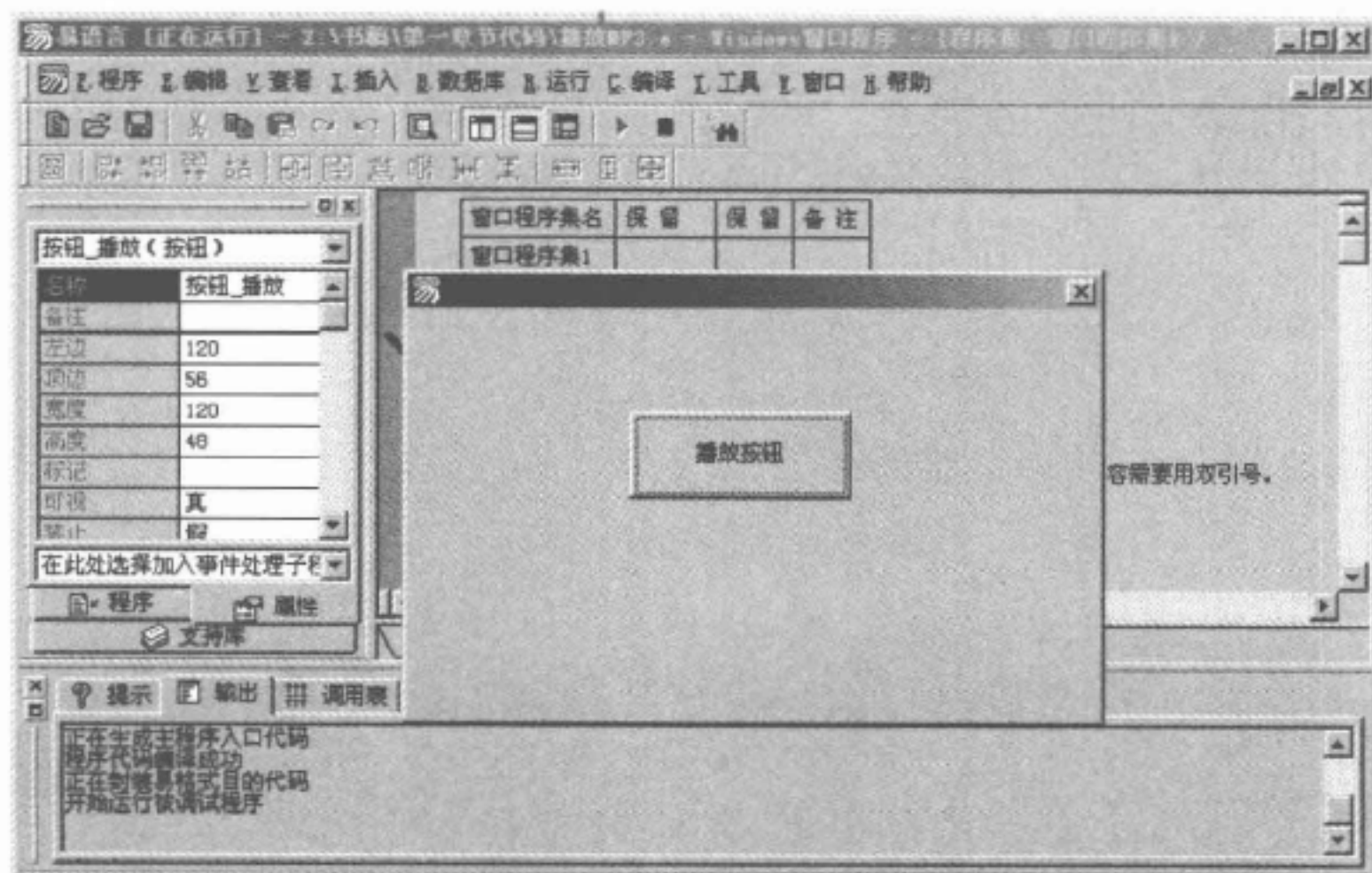


图1-55 运行结果





第五步：保存。

使用菜单“程序”→“保存”，保存这个文件。

思考：播放其他歌曲需要修改代码，那如何才能不修改而直接选择呢？

## 1.7 小结

易语言与其他编程语言基本上一样，但为中国用户定制了一些特色的功能，在此需要简单了解易语言的功能特色。如：易语言内置的输入法解决了中文输入慢的问题。

易语言的安装过程是较为简单的，只需跟着引导一步一步安装即可。大家可以自己尝试着下载、安装易语言最新版。

安装后可立即运行易语言，以熟悉易语言的界面，可试着调整易语言的一些配置。

编写易语言的代码需要有一些注意的事项，如可以使用首拼字母输入，可以展开参数输入，如何切换窗口等，通过学习以掌握集成化程序设计界面的使用。

大家要熟练地在窗口中添加组件和更改组件属性。最后练习本章后面介绍的进阶易程序，以了解易语言其他组件的添加，属性的修改，事件子程序的生成。

大家可进入易语言的论坛，若有问题可以在此论坛上提问，也可以在易语言资源网中查找、下载一些优秀的程序源码来加以分析、理解。







## 第二章 易语言编程基础

### 本章目标

在本章结束时，我们能够：

- 了解易语言中数据类型的分类
- 掌握易语言中变量的定义和赋值方法
- 掌握易语言中常量的定义和使用方法
- 掌握易语言中资源的添加和使用方法
- 了解易语言中运算符的分类和优先级
- 了解算术表达式的应用
- 熟练地编写和调用子程序





## 2.1 变量与常量

### 2.1.1 数据类型

#### 2.1.1.1 了解数据类型

数据：是指能够输入到计算机中，并能够被计算机识别和加工处理的符号的集合，是程序处理的最小对象。如：数值、字符、图形、图像和声音等都是数据。数据在程序中以常量或变量的方式被引用，不同的数据特点有不同的存储要求和处理算法。

上小学时，我们学过自然数、小数、分数等，其中，1、2、3这样的数叫自然数；-2、-1、0、1、2等的数叫整数；1.1、1.3、1.4、1.66这样的数叫小数。把数据进行这样的区分，在电脑语言里被称做数据类型。

易语言的数据类型从数据结构来区分，可分为基本数据类型、特殊数据类型。

基本数据类型有6种，包括数值型、逻辑型、日期时间型、文本型、字节集型、子程序指针型。

数值型数据又包括了字节型、短整数型、整数型、长整数型、小数型、双精度小数型。数值型数据都是由0~9数字和小数点组成的一个数值。这些类型代表的数值范围，及计算机内存存储的长度，具体可参照表2-1。

表2-1 数据类型的长度和存储的值域

数据类型名称	位长度	占用字节数	取值范围
字节型	8	1	0 ~ 255
短整数型	16	2	-32,768 ~ 32,767
整数型	32	4	-2,147,483,648 ~ 2,147,483,647
长整数型	64	8	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
小数型	32	4	-3.4E38 ~ 3.4E38 (7位小数)
双精度小数型	64	8	-1.7E308 ~ 1.7E308 (15位小数)
逻辑型	32	4	“真”或“假”
日期时间型	64	8	100年1月1日~9999年12月31日
子程序指针	32	4	长度为4个字节。此数据类型的数据用来与外部程序或操作系统API进行交互，是一个子程序在内存中的地址
文本型			由以字节0结束的一系列字符组成
字节集			一段字节型数据串

**注解：**各种类型的数据都在内存中占用一定的存储空间。字节是计算机中数据处理的基本单位。计算机中以字节为单位存储和解释信息，规定1个字节由8个二进制位构成，即1个字节等于8个比特（1Byte=8bit）。数据类型所占字节数越多，所能够容纳数据容量就越大。





2.1.1.2 数据类型应用

数据类型可以用来描述不同变量的类型或组件属性的类型。通过下面对变量和返回值定义数据类型来具体了解一下。

1) 定义变量的数据类型

在声明一个变量时要定义其具体的数据类型，只需要在新建的变量的类型栏上按下空格键，选择下拉列表中欲定义的数据类型即可。

例如：对变量名为“日期时间”的变量定义其数据类型。

首先定义一个变量名为“日期时间”的变量，然后为该变量定义数据类型，如图2-1所示，在变量名“日期时间”后的“类型”处，按下空格键，就会弹出数据类型下拉列表，在列表中选择欲定义的数据类型即可。

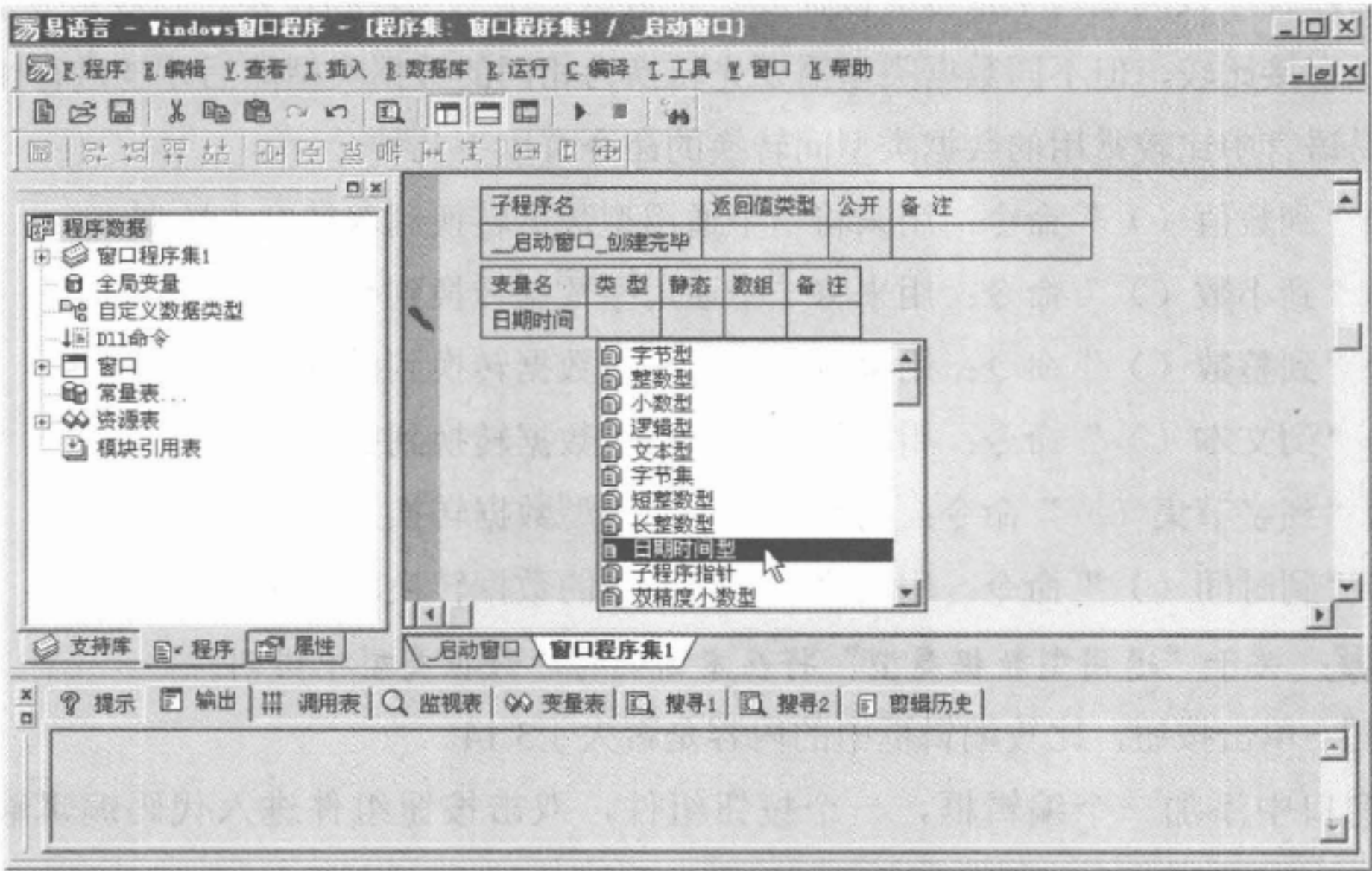


图2-1 选择数据类型

也可用首拼或英文在其位置直接输入，如图2-2所示的时间日期型即输入rqsjx或用英文输入date。

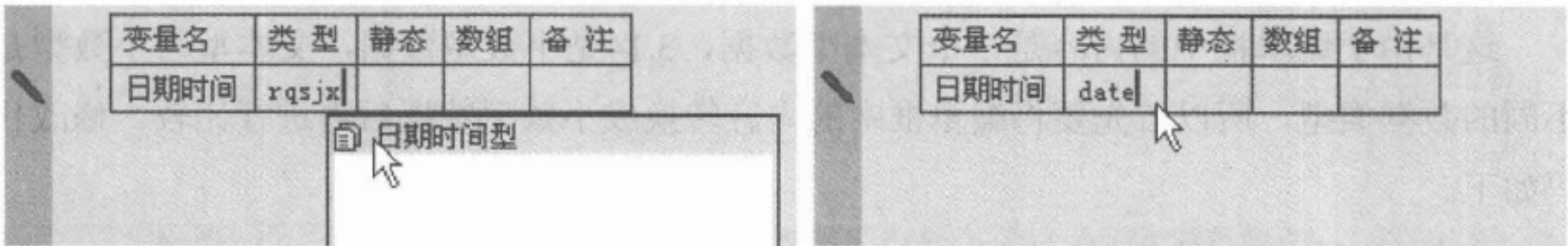


图2-2 输入数据类型

2) 定义子程序返回值数据类型

子程序如果具有返回值，我们也要相应的定义返回值的类型。在“返回值类型”处输入或选择相应的数据类型即可，方法同给变量定义数据类型。如：





子程序名	返回值类型	公开	备注		
加减运算	整数型				
参数名	类型	参考	可空	数组	备注
被运算数	整数型				
运算数	整数型				
运行的运算	整数型		✓		

--- 如果 (是否为空 (运行的运算))  
--- 返回 (被运算数 + 运算数)  
--- 返回 (被运算数 - 运算数)

### 3) 数据类型的转换

在实际程序代码编写中很多时候需要进行数据类型的比较和转换。同数据类型之间的数据可以直接比较,但不同数据类型需要先转换为相同的数据类型然后才能进行比较。

在易语言中比较常用的数据类型间转换的命令有以下几种。

- (1) “到数值 ()”命令:用来将一个通用型数据转换到双精度小数型。
- (2) “到小数 ()”命令:用来将一个通用型数据转换到小数型。
- (3) “到整数 ()”命令:用来将一个通用型数据转换到整数型。
- (4) “到文本 ()”命令:用来将一个通用型数据转换到文本型。
- (5) “到字节集 ()”命令:用来将一个通用型数据转换到字节集型。
- (6) “到时间 ()”命令:用来将一个文本型的数据转换成日期时间型。

**注解:** 关于“通用型数据类型”将在本章的特殊数据类型中讲到。

**【例】** 单击按钮,比较编辑框中的内容是否大于3.14。

在窗口中添加一个编辑框,一个按钮组件,双击按钮组件进入代码编辑输入如下代码。

信息框 (编辑框1.内容 > 3.14, 0,)

按F5键运行程序,发现在输出面板中会提示错误 (10044)。

错误(10044): 不能将“文本型”数据转换到“双精度小数型”数据。

这是由于编辑框中的内容是一个文本型数据,3.14是小数型数据,文本型与小数型是不同的数据类型,所以首先要将编辑框中的内容转换成小数型数据后再进行比较。修改代码如下。

信息框 (到小数 (编辑框1.内容) > 3.14, 0,)

程序运行后在编辑框中输入一个数值进行比较。如:78,然后点击按钮会弹出一个信息为“真”的信息框。

修改编辑框中的数值为1.2,点击按钮会弹出一个信息为“假”的信息框,如图2-3所示。



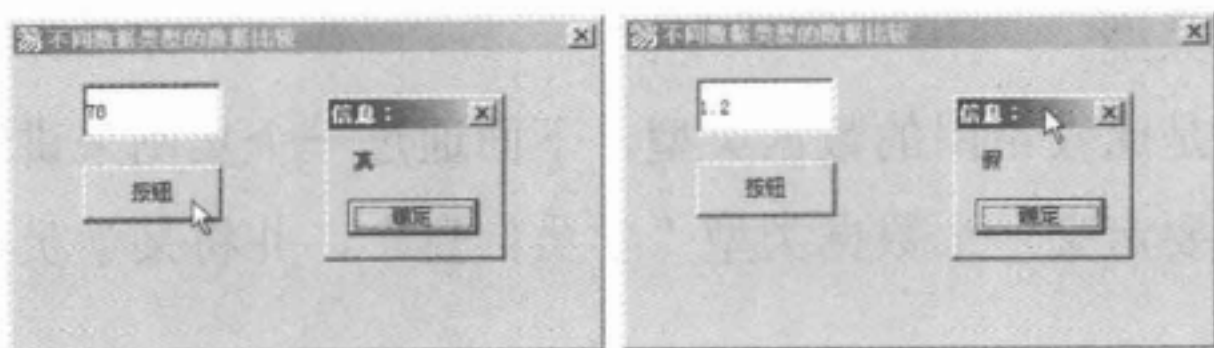


图2-3 比较数值大小

**【例】** 单击按钮，比较编辑框中的内容与所选日期的大小。

在窗口中添加一个编辑框组件，一个日期框组件和一个按钮组件，将编辑框内容改为：“2009年10月14日”或其他日期，双击按钮组件进入代码编辑输入如下代码。

信息框(到时间(编辑框1.内容) > 日期框1.今天, 0,)

由于编辑框1内的数据为文本型，所以要与日期框1.今天（日期时间型）比较的时候，就要先使用“到时间（）”命令将文本“2009年10月14日”转换为日期时间型，然后再比较。

如果不进行“到时间（）命令”的转化直接比较，

信息框(编辑框1.内容 > 日期框1.今天, 0,)

在输出面板中会提示错误（10044）。

错误(10044): 不能将“文本型”数据转换到“日期时间型”数据。

程序运行后，改变日期框的日期，点击按钮后如图2-4所示。

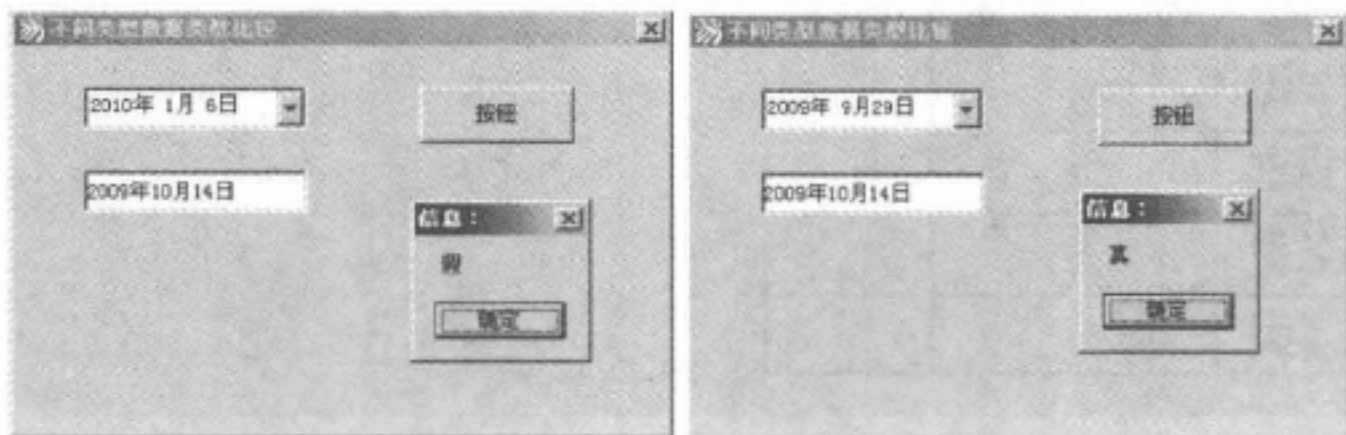


图2-4 比较日期大小

#### 4) 特殊数据类型

特殊数据类型是指除易语言基本数据类型之外的其他数据类型。包括通用型数据类型、库定义数据类型、自定义数据类型、内部组件数据类型。

**通用型数据类型：**通用型数据类型仅在系统内部使用，它能够匹配所有的系统基本数据类型、库定义数据类型、自定义数据类型。

**库定义数据类型：**库定义数据类型由易语言支持库提供，用户在程序中可以直接使用，就如同是系统基本数据类型一样。

**自定义数据类型：**用户可以随时在程序中自行定义新的数据类型。但自定义数据类型时需要设置数据类型的名称及其成员，其中数据类型成员各属性的设置方法等同于变量设置方法。

**内部组件数据类型：**在易语言中，每一种内部组件都可以作为一种数据类型来使用。



## (1) 自定义数据类型

自定义数据类型是比较常用的数据类型，下面通过一个实例来讲解如何定义一个自定义数据类型。例如：要定义一个数据类型“学员信息”，并将某学员的信息做个记录。方法如下。

新建一个易程序，在程序面板“自定义数据类型”处单击右键，选择“插入新数据类型”，如图2-5所示。

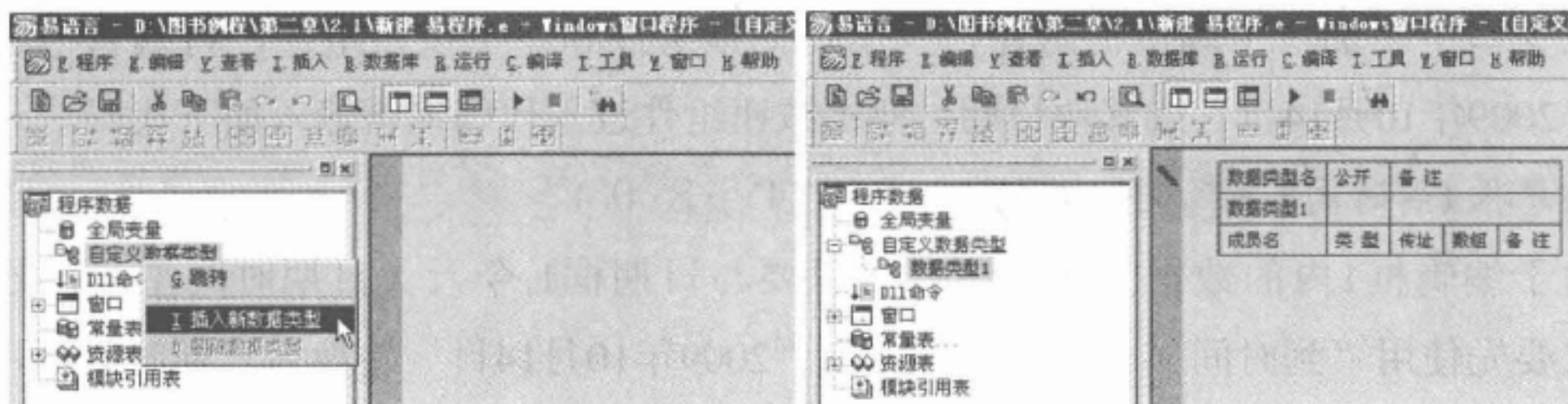


图2-5 新建自定义数据类型

修改数据类型名为“学员信息”，在“成员名”处回车，加入4个成员“姓名”、“性别”、“年龄”、“班级”，数据类型分别为“文本型”、“逻辑型”、“整数型”、“文本型”。

数据类型名	公开	备注		
学员信息				
成员名	类型	传址	数组	备注
姓名	文本型			
性别	逻辑型			
年龄	整数型			
班级	文本型			

在启动窗口中添加一个按钮，双击该按钮，在“按钮1”的单击事件子程序中，新建一个变量“学员1”（Ctrl+L），数据类型设置为“学员信息”类型。并设置学员1的相关信息，详细代码如下。

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
学员1	学员信息			

学员1.姓名 = “王小二”  
 学员1.性别 = 真  
 学员1.年龄 = 12  
 学员1.班级 = “一年二班”

这样，在单击“按钮1”时，一个学员的信息就生成了，在程序的其他地方就可以调用这个学员的信息了。



## (2) 内部组件数据类型

在易语言中，每一种内部组件也可以作为一种数据类型来使用。这些数据类型具有组件的特征，如属性、命令等。下面通过一个实例来具体讲解：

**【例】** 新建一个标签组件“标签1”，一个按钮组件“按钮1”，如图2-6所示。

双击“按钮1”进入代码编辑，新建一个变量“变量1”（CTRL+L），定义数据类型为“标签”，并将标签1赋值给变量1，这样变量1就相当于标签1的一个别名，所有对变量1的操作都会影响标签1；更改变量1的“左边”、“顶边”、“标题”。具体代码如下。

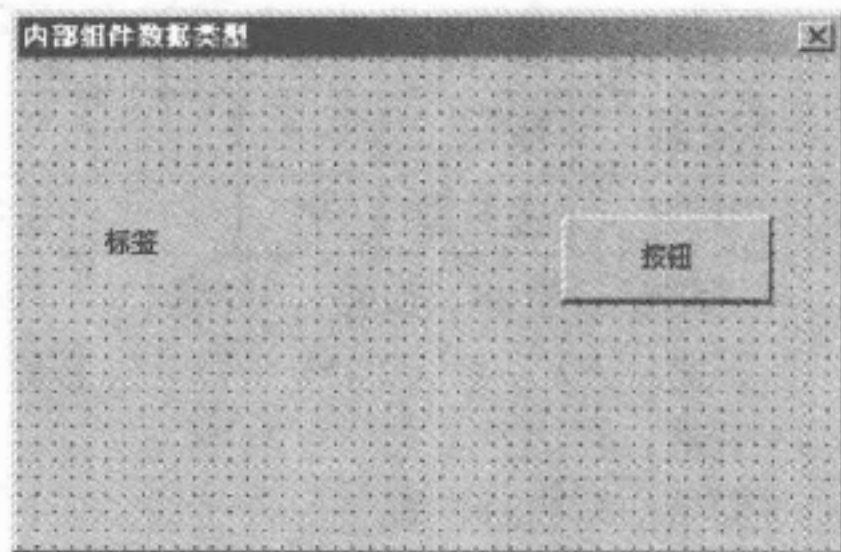


图2-6 内部组件示例界面

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	标签			

```
变量1 = 标签1
变量1.左边 = 100
变量1.顶边 = 100
变量1.标题 = “我是一个标签”
```

运行效果如图2-7所示。

## (3) 动态添加组件

利用内部组件可以作为数据类型的特点，易语言中可以动态添加组件。要用到一个“复制窗口组件（）”命令。命令原型为。

<逻辑型> 复制窗口组件（通用型 欲被复制的窗口组件，通用型变量 存放新创建组件的变量）

**注解：** 被复制的组件必须是窗口中已经存在的组件。

下面通过一个实例来具体讲解：

**【例】** 在窗体上添加一个按钮，然后动态添加其他按钮，实现每单击按钮一次，就生成一个新的按钮，如图2-8所示。

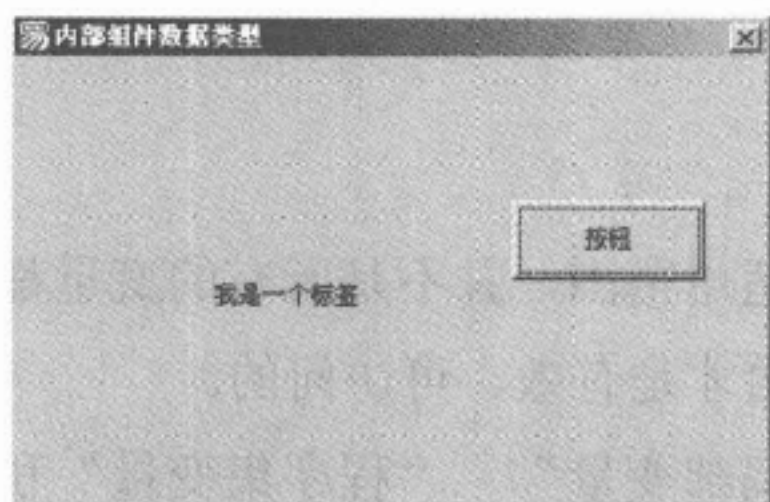


图2-7 使用内部组件数据类型示例

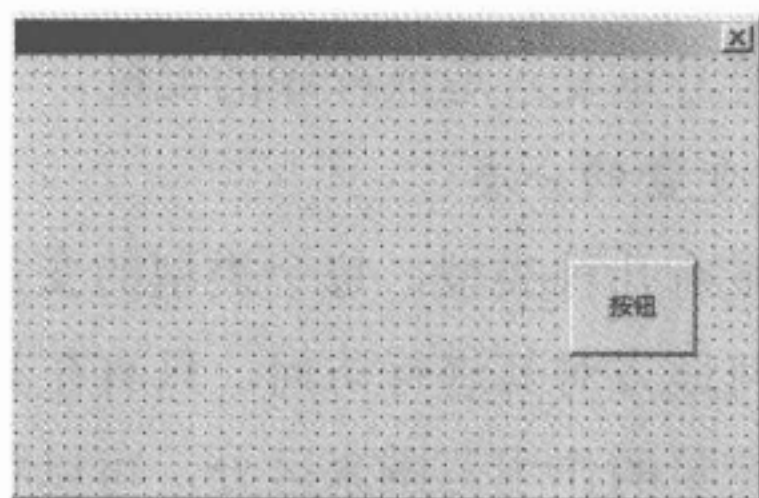


图2-8 动态添加组件示例界面



在窗体上添加一个按钮“按钮1”，双击“按钮1”进入代码编辑，编写如下代码。

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	按钮			
个数	整数型	✓		

复制窗口组件 (按钮1, 变量1)  
变量1.左边 = 50 + 个数 × 按钮1.宽度  
变量1.顶边 = 10  
变量1.可视 = 真  
个数 = 个数 + 1

运行程序，多次点击“按钮1”，效果如图2-9所示。

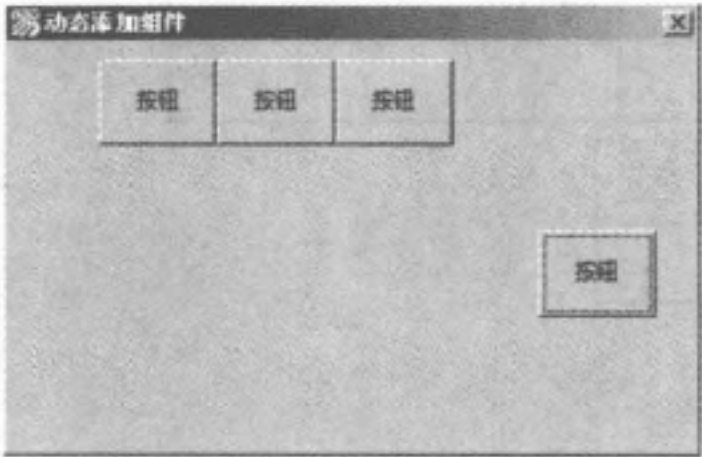


图2-9 动态添加组件示例

2.1.2 变量

变量在程序中的应用是非常广泛的。可以把变量理解成一个容纳物品的容器，只是这个变量容纳的是各种可变的数据。如衣服的口袋，可以放入东西也可以取出东西。变量也是一样的，可以提取变量中的数据，也可以改变变量中的数据。

变量可以任意定义名称，不过在具体程序开发中会根据需要给变量起一个有实际意义的名字，这样的目的是为了便于程序的开发、维护及相互交流。在变量使用很多的时候，定义变量名尤为重要。养成一个良好的定义名称的习惯，对以后的程序开发会有很大帮助。

虽然变量可以任意定义名称，但定义变量名时也要注意：首字符不可以是数字，并且变量名中除“\_”以外，不可以使用其他的符号和标点。主要原因是为了避免与程序中的数值和符号重复，造成程序的混乱。

1) 变量的分类

变量有几种类型，每个类型的变量都有自己的适用范围，并不是所有的变量都能在程序的任何地方对其进行访问，只有在其作用域范围内才是有效、可访问的。

从变量的作用范围来区分，可以将变量分为“局部变量”、“程序集变量”和“全局变量”。



(1) 局部变量：只能在其所在的子程序中才能被调用的变量，其他子程序都无法调用。因为只有子程序被调用的时候，其变量才占用系统的内存，当子程序执行结束后，其变量所占内存空间将被系统收回，因此局部变量也是非常节省系统内存的。

(2) 程序集变量：一般情况下仅在本程序集中被调用。若在其他窗口程序集中调用，则需要在变量名前加程序集所对应的“窗口名称”前缀。如：

信息框 ( \_启动窗口. 变量\_程序集变量, 0, )

程序集变量所在的程序集中的所有子程序，都可以自由访问程序集变量，因此多个子程序都需要访问的数据，可以使用程序集变量来存储。

(3) 全局变量：在程序运行后，所有程序集内子程序都可以使用的变量。全局变量也是覆盖范围最大的变量。这种变量在程序运行后即占用内存空间，在程序运行结束后才从内存中清除，所以会长时间占用系统资源，因此全局变量建议根据程序的实际情况使用。

变量还可以从变量的属性来区分，分为“静态变量”和“数组变量”。

(1) 静态变量：静止存在的局部变量。当所处子程序退出时，静态局部变量能够保留住现行内容以供下次继续使用；但非静态的变量就不能，下次进入子程序时它将被重新初始化。局部变量如果不设置“静态”属性，子程序执行完毕后，将清空该子程序中的所有非静态局部变量；局部变量如果设置了“静态”属性，在子程序执行完毕后也不会被清空，当子程序再次被调用时，静态变量的值仍保持上次被调用时的状态。静态变量在子程序第一次被调用时分配内存；在程序结束时销毁。

(2) 数组变量：即可以存放一组数据的变量。数组变量中的每个成员都拥有独立的存储单元，可以单独调用和赋值。数组变量可以看做是多个非数组的变量组成的。数组变量又分为“单维数组变量”和“多维数组变量”。

单维数组变量，成员的表示形式为：

数组变量名 [数组成员下标]

如：变量 [3]，表示该数组中的第3个成员。

**注解：**数组下标是从1开始的，所以不存在“变量[0]”这个数组成员。

多维数组变量可以理解成一个特殊的单维数组，它的成员也可以看成是多个单维数组。多维数组成员的表示形式为：

多维数组变量名 [数组成员下标] [数组成员下标]

如：变量 [1] [2]，意思是一个二维变量中的第二个成员。

## 2) 定义变量

各种变量是如何定义的呢？这里将详细介绍一下。

(1) 定义局部变量：可以在需要定义变量的子程序处，使用Ctrl+L新建“局部变量”，或者在易语言的“插入”菜单中选择“局部变量”来插入一个局部变量，如图2-10所示。



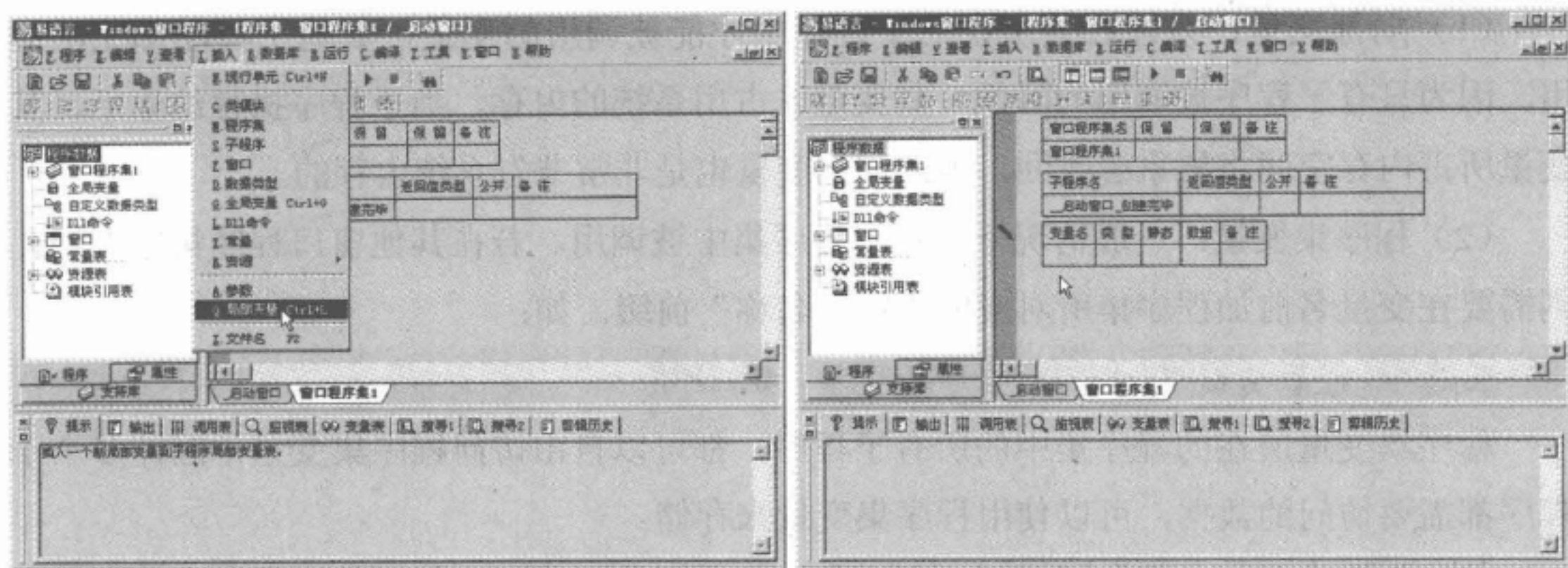


图2-10 通过菜单插入新的局部变量

在变量名处输入变量的名称，并选择相应的数据类型即可，如图2-11所示。

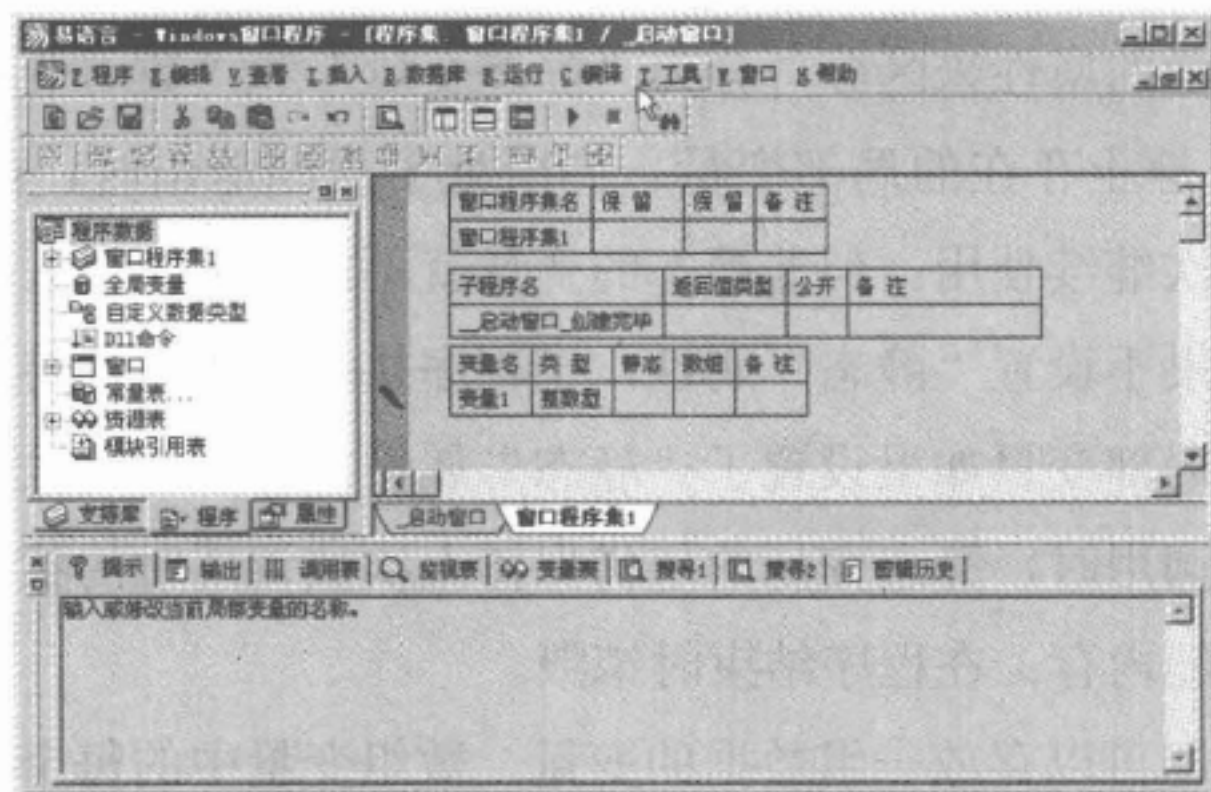


图2-11 通过菜单插入新的局部变量

(2) 定义程序集变量：将光标停留在代码编辑区“窗口程序集”中任意位置，按回车键，生成一个新行，如图2-12所示。

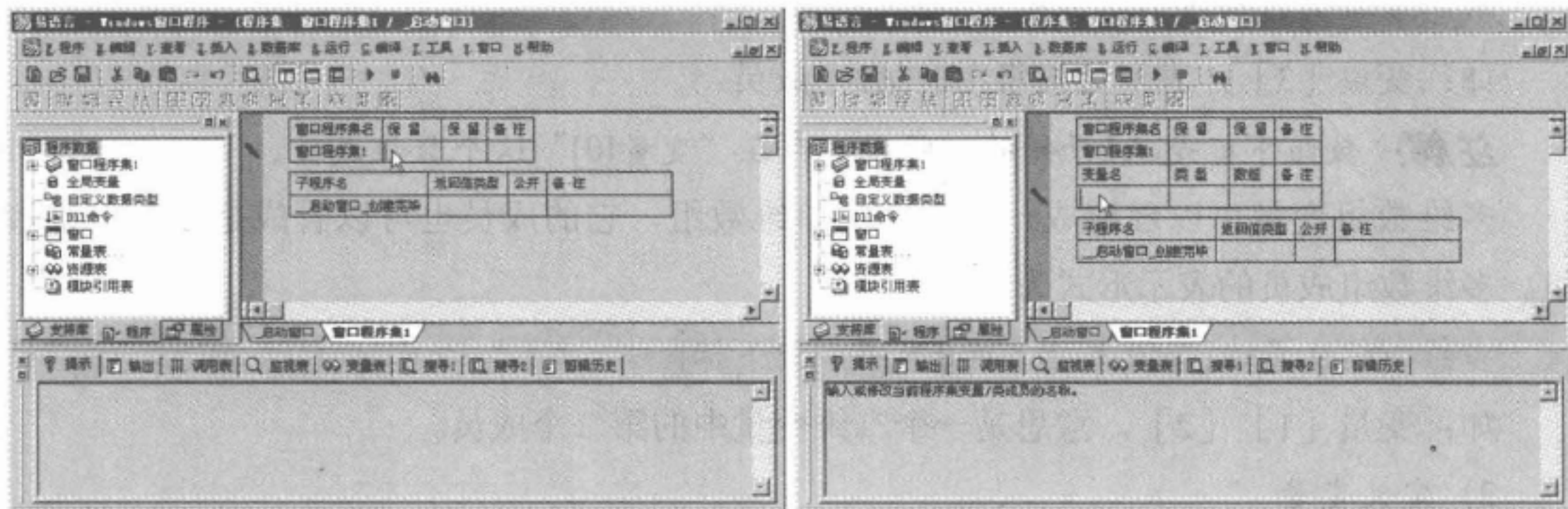


图2-12 插入新的程序集行

在变量名处输入变量的名称，并选择相应的数据类型即可，显示效果如图2-13所示。

(3) 定义全局变量：定义全局变量可以用Ctrl+G，新建一个“全局变量”。还可以在易语言的“插入”菜单中选择“全局变量”来插入全局变量，如图2-14所示。





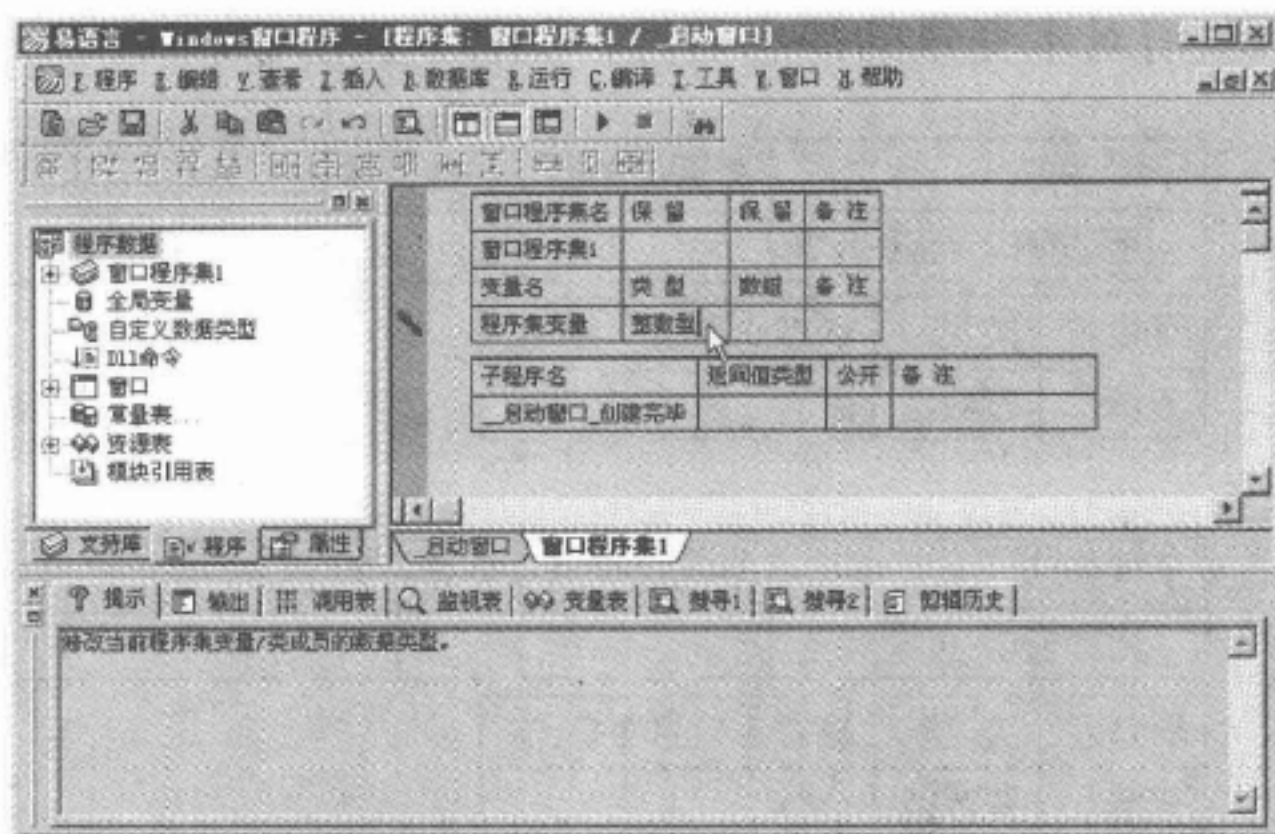


图2-13 编辑新的程序集变量

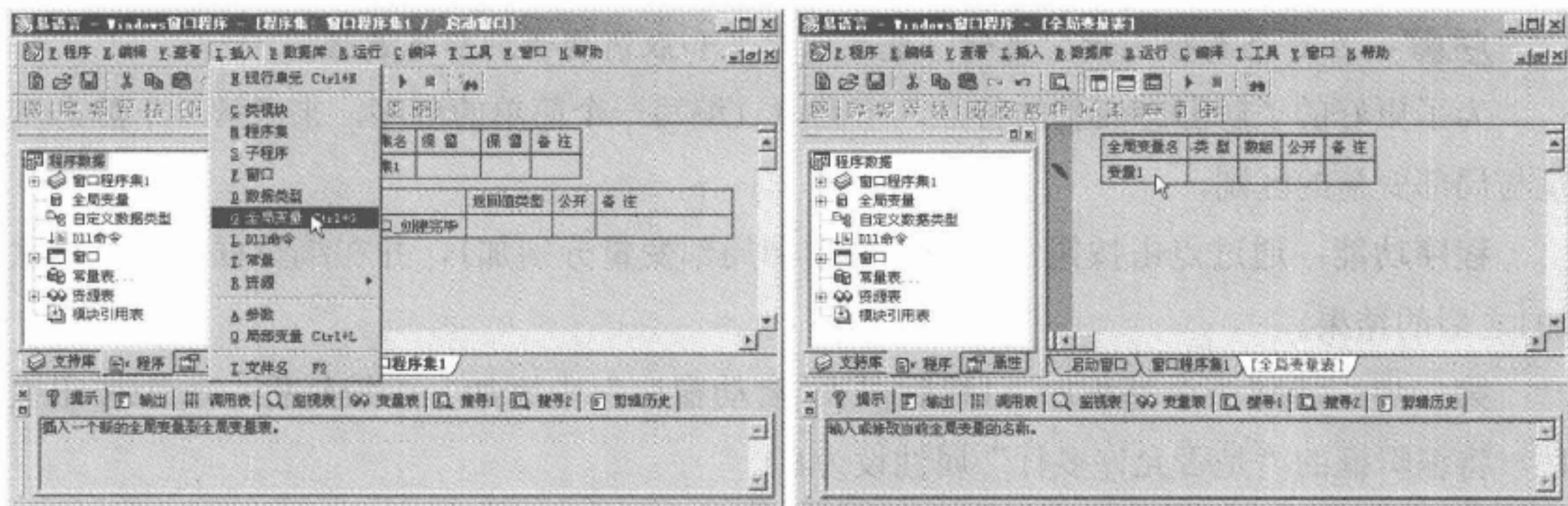


图2-14 插入新的全局变量行

在“全局变量名”处输入修改全局变量的名称，并选择相应的数据类型，效果如图2-15所示。

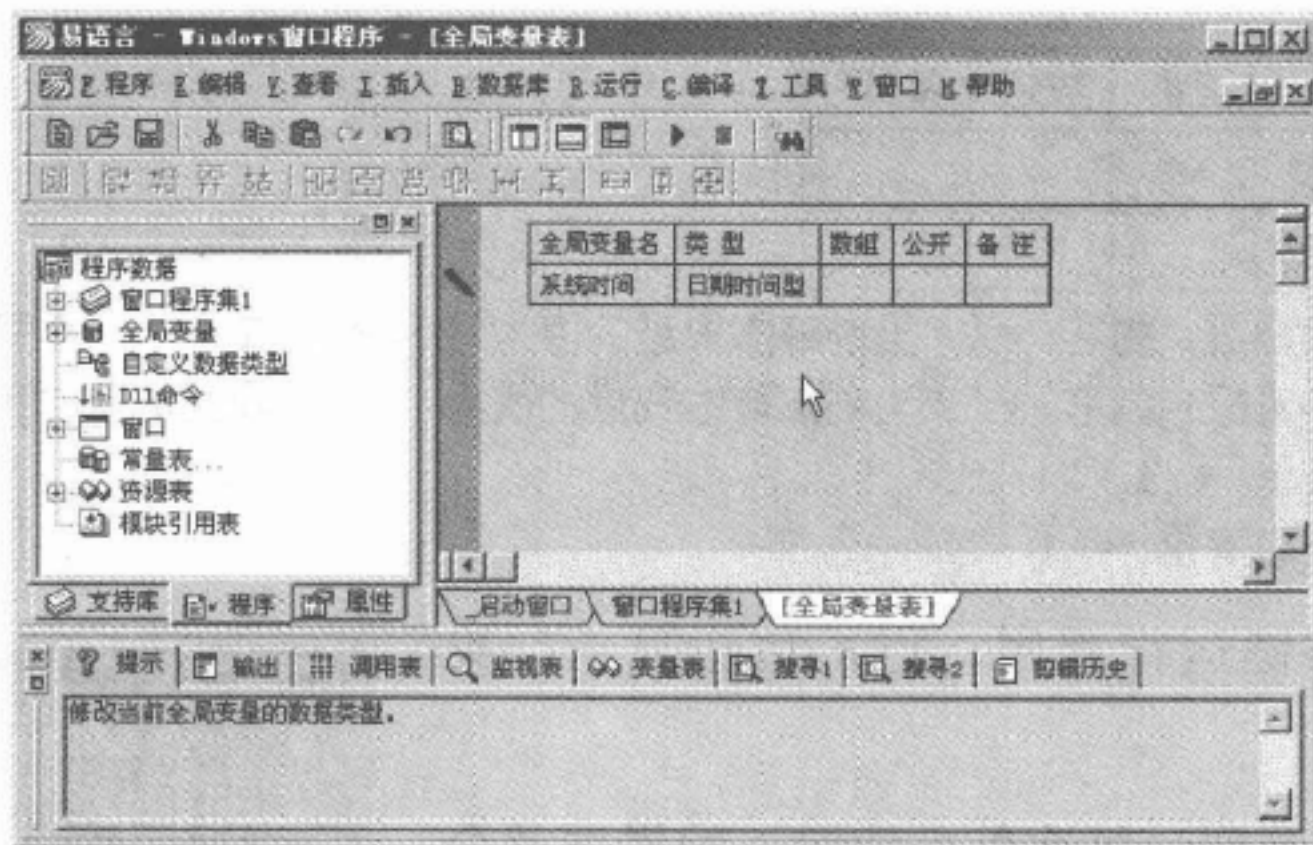


图2-15 编辑新的全局变量

(4) 定义静态变量：静态变量的定义方法是，首先和前面定义局部变量的方法相似，定义一个局部变量，然后将光标停留在“静态”处，单击鼠标左键，可以定义或取消静态属性。当静态属性上出现“√”后，即表示这个定义的变量是静态变量，当再次点击将



“√”去掉，即表示取消对本变量的静态属性，如图2-16所示。

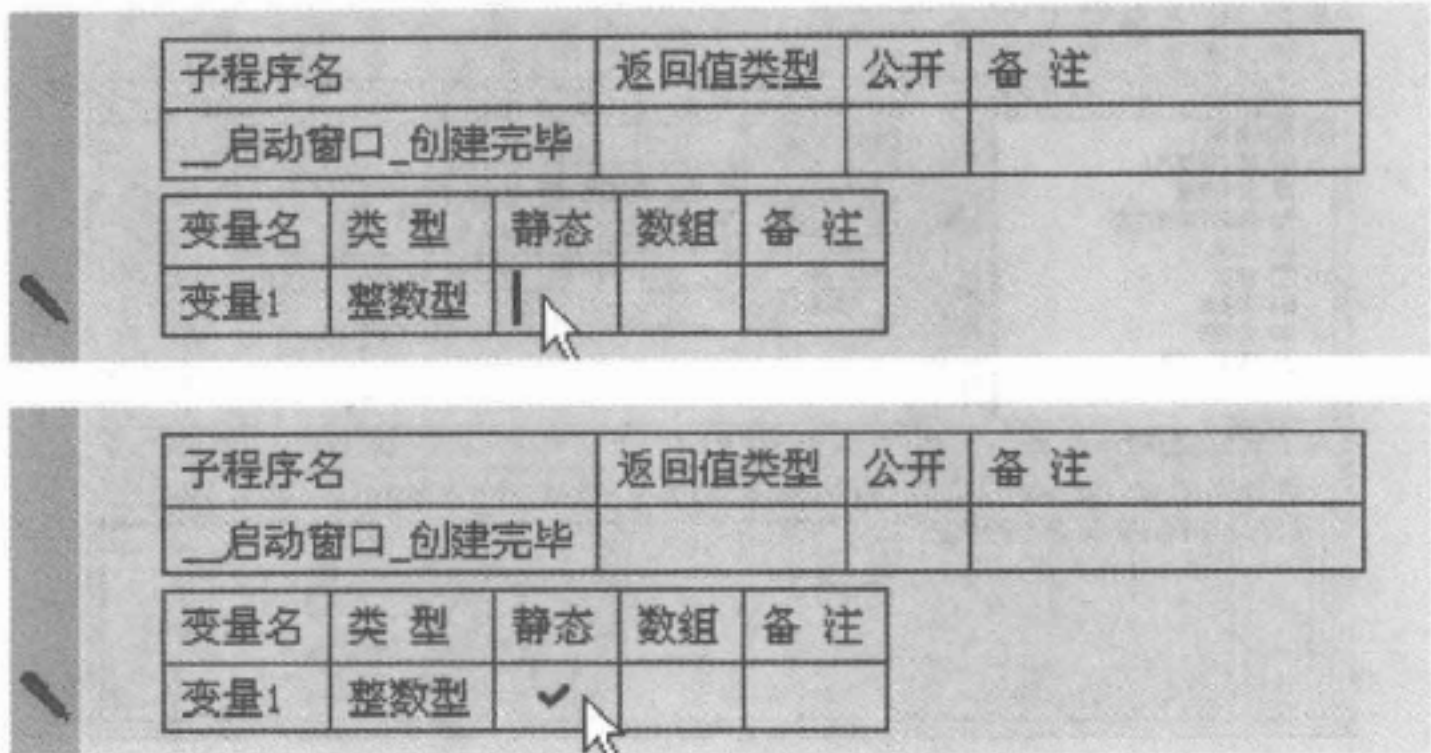


图2-16 定义静态变量

**注解：**在静态属性上按空格键，也可以定义和取消变量的静态属性。

为了更好的了解静态变量的特点，下面通过编写一个简单的程序，来了解一下静态变量与局部变量的区别。

**程序功能：**通过点击按钮，使静态变量和局部变量分别加1，并在编辑框中显示每次加1之后的结果。

**第一步：**新建一个易程序，然后在“\_启动窗口”中添加一个编辑框、一个按钮组件。将编辑框的“是否允许多行”属性设为真。

**第二步：**双击按钮组件，在按钮的单击事件中，新建静态变量与非静态变量各一个，并输入如下代码。

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
静态变量	整数型	✓		
非静态变量	整数型			

```
编辑框1.内容 = 编辑框1.内容 + “当前静态变量的值为:” + 到文本(静态变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “当前非静态变量的值为:” + 到文本(非静态变量) + #换行符  
静态变量 = 静态变量 + 1  
非静态变量 = 非静态变量 + 1
```

**第三步：**按F5键运行程序，点击按钮显示结果如图2-17所示。

从程序中可以看到，每次单击按钮，编辑框就会显示2行内容，其中第二行的数字是不变的，显示的是非静态变量中的数值；而第一行的数字每次都递增1，显示的是静态变量中的数字。这是因为：每次点击按钮，静态变量和非静态变量都会加上1。非静态

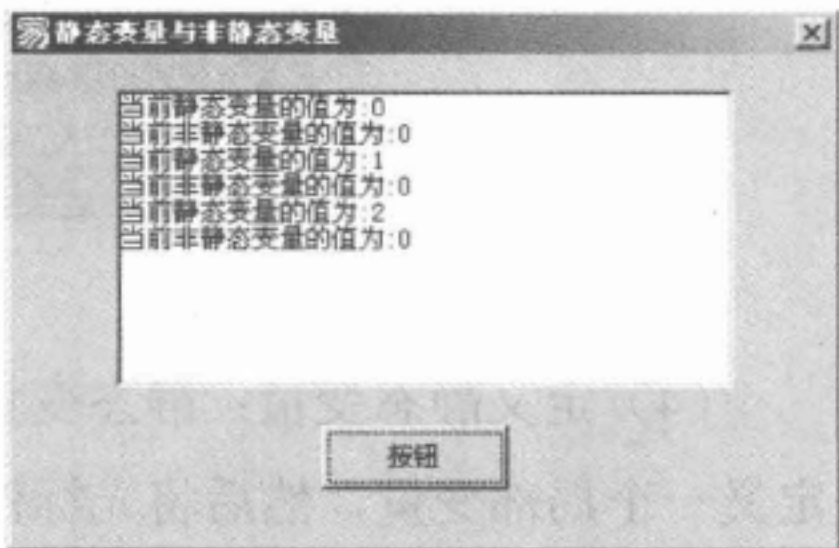


图2-17 静态变量与非静态变量



变量的值每次点击按钮后，都会重新初始化，所以它的值始终是0，而静态变量能够保留住当前值，会逐次加1。

(5) 定义数组变量：数组变量的定义方法和前面定义静态变量的方法相似。

定义一个变量，将该变量命名为“数组”，然后将光标停留这一行后面的“数组”处，输入要定义的成员数即可，如图2-18所示。

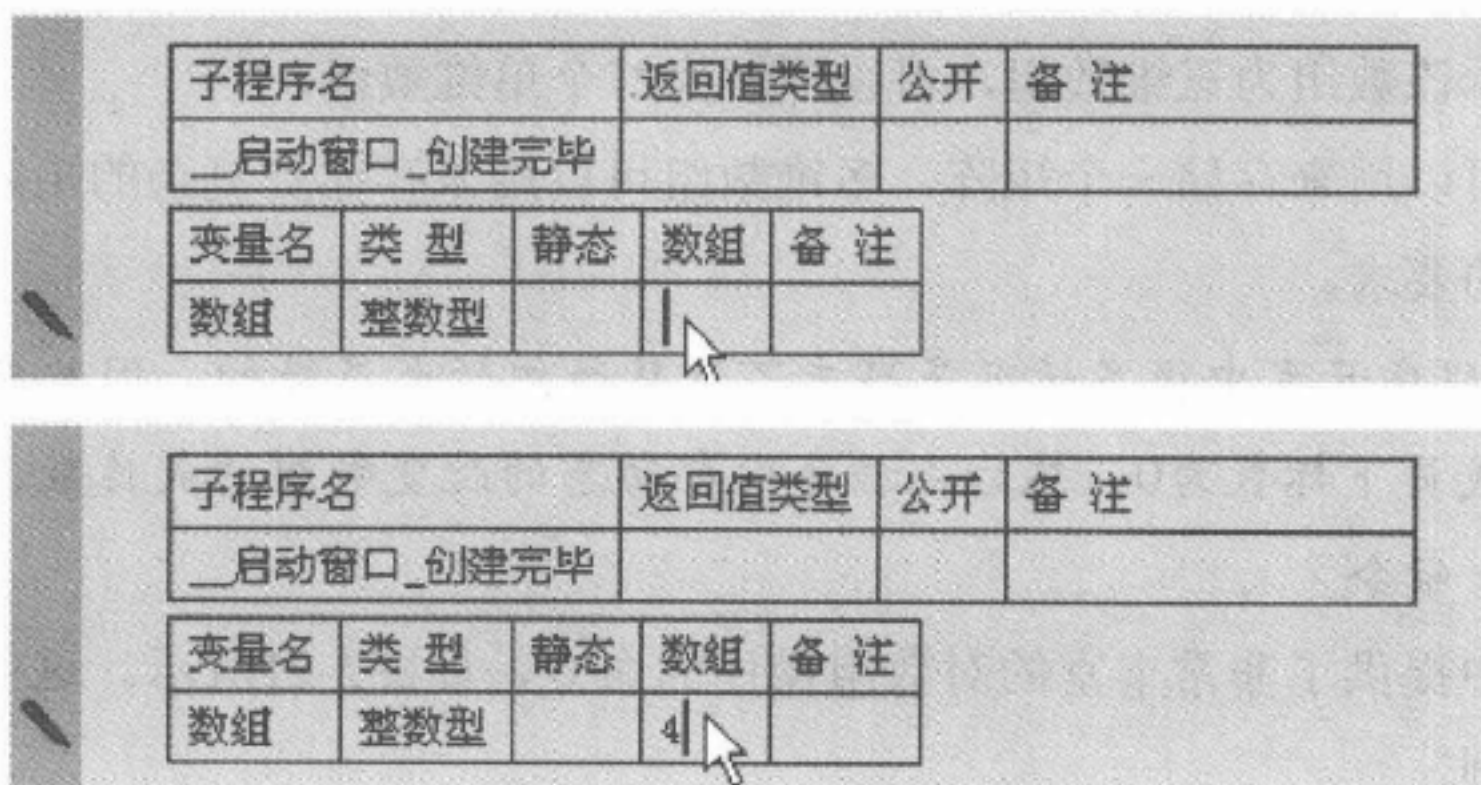


图2-18 定义数组变量

这样“数组”就有数组[1]、数组[2]、数组[3]、数组[4]四个数组成员。

如果在数组属性中输入类似“3, 3”，即将本变量“数组”定义为2维，如图2-19所示：

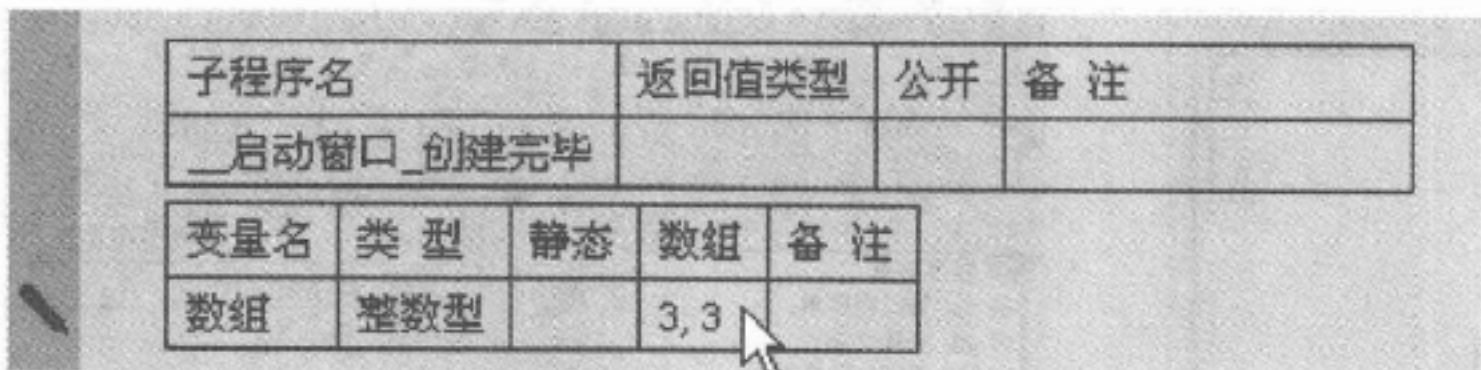


图2-19 定义二维数组变量

上图表示该数组为二维数组，可以看成是3个拥有3个成员的单维数组组成，这个数组中的成员按如下顺序排列：

数组[1][1]、数组[1][2]、数组[1][3]；

数组[2][1]、数组[2][2]、数组[2][3]；

数组[3][1]、数组[3][2]、数组[3][3]。

多维数组成员可以以单维数组的表示方法来访问，如以上的二维数组中9个成员，按成员顺序排列也可以表示为：

数组[1]、数组[2]、数组[3]；

数组[4]、数组[5]、数组[6]；

数组[7]、数组[8]、数组[9]。

按照定义单维数组变量与二维数组变量的方式来定义一个三维数组，如图2-20所示。



子程序名		返回值类型	公开	备注
_启动窗口_创建完毕				

变量名	类型	静态	数组	备注
数组	整数型		3, 3, 3	

图2-20 定义三维数组变量

以上这表示该数组为三维数组，可以理解为27个单维数组。

二维数组可以用来存储一个矩阵，多维数组可以用来存储更复杂的矩阵，满足编程中遇到的更复杂的要求。

**注解：**在程序开发中很多情况是事先不清楚数组的成员数的，因此在定义数组时可以定义数组的成员下标数为0，然后在程序内部动态的改变数组的成员数，具体可以使用“重定义数组”命令。

在易语言中提供了非常丰富的对数组操作的命令，如图2-21所示。这些可以在支持库的帮助中查询到。

通过这些命令可以非常方便的管理和使用数组变量。数组变量的操作类命令非常重要，本书将在第三章的数组操作命令中详细的介绍数组操作的相关知识。

在程序代码中引用了没有定义的变量，则该行代码不会通过编译。按下F5键运行程序，则状态夹输出面板中会有相应的错误提示，如图2-22所示。

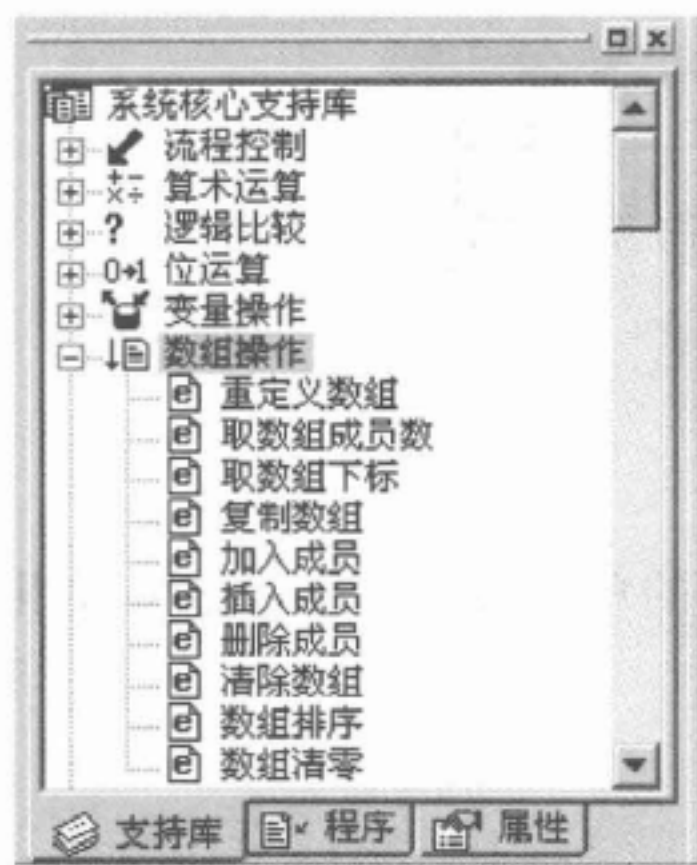


图2-21 数组操作命令

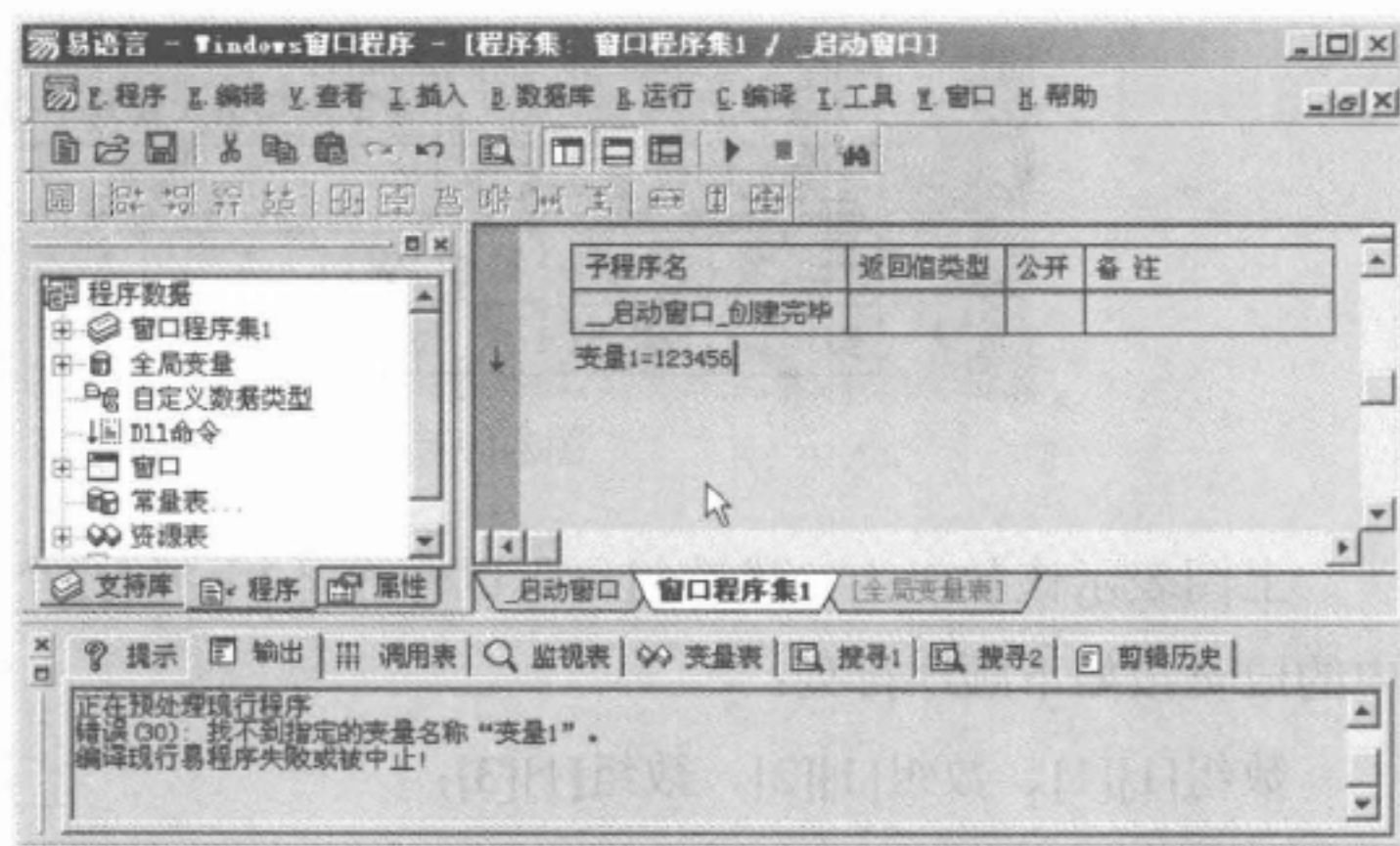


图2-22 变量未定义的错误

有时候为了程序代码快速开发，可以在编写代码时使用一些未定义的变量，但一定要在输入代码后按回车来补建变量，或者按F5键运行程序，按照错误提示将未定义的变量定义。

### 3) 变量的初始值

变量在没有赋值前，系统会为其赋予一个初始值。每一种数据类型的变量初始值都有所不同，见表2-2。



表2-2 变量的初始值

变量类型	变量初始值	初始值在代码中的表示方法
数值型	0	0
逻辑型	假	假
日期时间型	1899年12月30日	[1899年12月30日]
文本型	空文本	“ ”
字节集型	空字节集	{ }

为了方便理解可以通过一个小例程来具体看一下。

【例】新建一个易程序，添加一个编辑框和一个按钮组件。将编辑框的“是否允许多行”属性设为真。双击按钮组件，在“\_按钮1\_被单击”的子程序中新建十个变量，具体代码如下：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
字节型变量	字节型			
短整型变量	短整型			
整型变量	整型			
长整型变量	长整型			
小数型变量	小数型			
双精度小数型变量	双精度小数型			
逻辑型变量	逻辑型			
日期时间型变量	日期时间型			
文本型变量	文本型			
字节集变量	字节集			

编辑框1.内容 = 编辑框1.内容 + “字节型变量初始值:” + 到文本(字节型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “短整型变量初始值:” + 到文本(短整型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “整型变量初始值:” + 到文本(整型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “长整型变量初始值:” + 到文本(长整型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “小数型变量初始值:” + 到文本(小数型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “双精度小数型变量初始值:” + 到文本(双精度小数型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “逻辑型变量初始值:” + 到文本(逻辑型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “日期时间型变量初始值:” + 到文本(日期时间型变量) + #换行符  
编辑框1.内容 = 编辑框1.内容 + “文本型变量初始值:” + 文本型变量 + #换行符  
编辑框1.内容 = 编辑框1.内容 + “字节集变量初始值:” + 到文本(字节集变量) + #换行符

程序运行后，单击按钮，会看到如图2-23所示结果。

注解：其中文本型和字节集型变量初始值是空值，“到文本()”后不显示。

4) 变量的赋值方法

给变量赋值可以使用“赋值”命令或“连续赋值( )”命令。

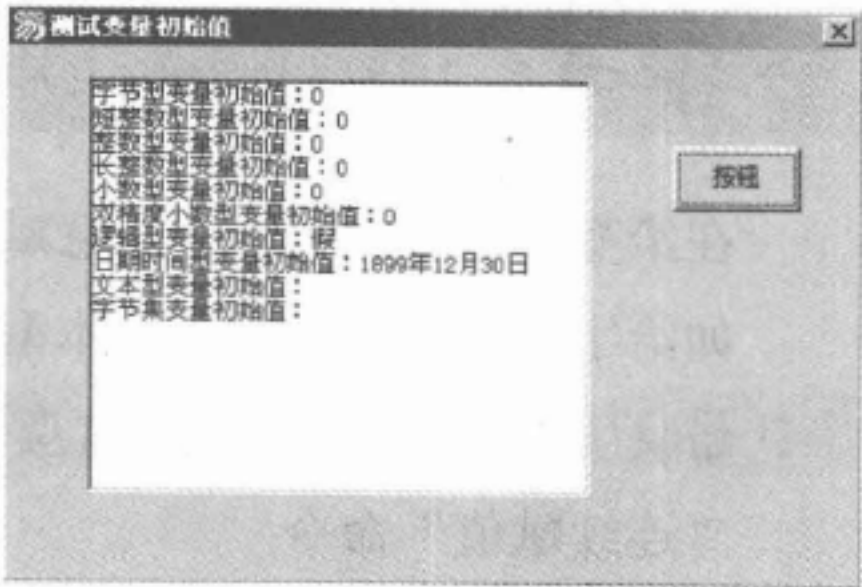


图2-23 测试变量的初始值



## “赋值”命令

命令原型为：

<无返回值> 赋值 (通用型变量/变量数组 被赋值的变量或变量数组, 通用型数组/非数组 用作赋予的值或资源)

本命令运算符号为“=”。将指定的常数、常数集、常量、资源、对象或者变量赋予到指定的变量或变量数组中去。赋值运算符在所有运算符中优先级最低, 是一条语句中最后被执行的命令。

如给以下各种变量的赋值, 都是正确的赋值方式。

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
字节型变量	字节型			
短整型变量	短整型			
整型变量	整型			
长整型变量	长整型			
小数型变量	小数型			
双精度小数型变量	双精度小数型			
逻辑型变量	逻辑型			
日期时间型变量	日期时间型			
文本型变量	文本型			
字节集变量	字节集			

### 给变量赋值

字节型变量 = 255

短整型变量 = 32767

整型变量 = 65535

长整型变量 = 58736232332

小数型变量 = 3.14

双精度小数型变量 = 3.141592654

逻辑型变量 = 真

日期时间型变量 = [2009年11月26日]

文本型变量 = “易语言”

字节集变量 = { 192, 168, 1, 1 }

在给变量赋值的时候要注意赋给变量相应数据类型的数据, 否则会出现错误。

如给字节型变量赋值为文本型的“我爱易语言”, 则会提示错误。

错误(10044): 不能将“文本型”数据转换到“字节型”数据。

## “连续赋值”命令

命令原型为：



<无返回值> 连续赋值 (通用型数组/非数组 用作赋予的值或资源, 通用型变量/变量数组 被赋值的变量或变量数组, ...)

本命令将指定的常数、常数集、常量、资源、对象或者变量赋予到一系列变量或变量数组中去。本命令的被赋值变量可以是一个或多个。如:

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
变量1	整数型			
变量2	整数型			

» 连续赋值 (5, 变量1, 变量2)

执行后, 变量1和变量2的值均被改变为5。

给变量赋值的时候要注意变量的数据类型, 只要符合其数据类型的赋值规则即可。

**注解:** 任意数值类型的数据可以被写入到其他任意数值类型的变量中, 系统将自动进行转换。如将一个整数写入到短整数型变量中要注意不要超出变量的取值范围; 将小数型变量写入到整数型变量时, 会丢失小数点后的内容等, 因此使用时需要注意, 最好转换类型是一一对应。

在给一组相同类型的组件的属性赋予相同的值时, 使用连续赋值命令非常方便, 代码在以后的阅读中也易于理解, 常用来给一组组件赋初始化的值。

5) 数组变量的赋值

给数组变量赋值, 可以有多种方法, 大家可以根据需要选择合适的赋值方法。

(1) 使用“=”赋值

例如:

① 给一维数组2个成员赋值, 每个成员都为10。代码如下:

```
数组[1]=10
数组[2]=10
```

② 给二维数组赋值, 每个维有2个成员, 每个成员都为10。代码如下:

```
数组[1][1]=10
数组[1][2]=10
数组[2][1]=10
数组[2][2]=10
```

使用“=”赋值也可以一次性给多个数组变量赋值, 方法是使用一对大括号将要赋予的值括起来, 每个值都用“,”号隔开, 被隔开的值赋予数组中的对应位置的成员。

例如: 给一维数组2个成员赋值, 每个成员都为10。代码如下:

```
数组 = {10,10}
```



使用这种方法给成员很多的数组赋值尤为方便。

例如：给一个有10个成员的一维数组赋值，代码可以如下输入：

```
变量 = {1,2,3,4,5,6,7,8,9,10}
```

(2) 使用“连续赋值 ( )”命令

例如：给数组的三个成员赋值，可以使用如下方法：

```
连续赋值(10, 数组 [1], 数组 [2], 数组 [3])
```

执行后，数组 [1]、数组 [2]、数组 [3]的值均被改变为10。

(3) 直接用命令的返回值给数组赋值：有些命令的返回值就是一个数组，所以可以直接使用该返回值给数组赋值。赋值的时候首先要注意，根据命令返回数组的数据类型来给数组定义数据类型；也要注意数组的成员数是可变的，并且命令返回的数组成员数也不固定，所以可以定义接收返回值的数组成员数为0，当该数组接收了命令的返回值后，会自动定义成员数。

例如，“分割文本 ( )”命令的返回值，就是一个文本型的数组，在程序中可以表示为：

变量名	类 型	静态	数组	备 注
文本数组	文本型		0	

```
文本数组 = 分割文本 (“2009-11-12”, “-”, )
```

代码运行后，文本数组就有了3个成员，代表年月日，每个成员的值就是返回来的子文本。即文本数组[1]的值为“2009”，文本数组[2]的值为“11”，文本数组[3]的值为“12”。

数组是一个特殊的变量，它能一次声明很多相同类型的变量。当用户在程序设计过程中需要很多同种数据类型的变量时，定义一个数组就可以轻松解决这个难题，并且在使用时便于变量的管理，代码的阅读理解。

### 2.1.3 常量

计算机内一般以两种方式将程序中用到的数据存储在内存中，其中一种方式是在上一节讲过的变量的方式，而另一种就是使用常量的方式。常量与变量不同，在程序中使用它们时有一套专用的处理方法。

常量是一个固定的值，可用于定义在程序内多个位置使用的值，并且此值在正常情况下不会更改。使用常量而不直接使用常量值能让代码更加便于理解，此外，如果确实需要更改通过常量定义的值，则只需在一个位置（常量声明处）更改该值，而不需在不同位置都更改该值。

常量只能在程序设计时定义，在程序执行过程中是不能修改其值。

常量的类型可以为数值型、文本型、逻辑型、日期时间型和长文本型。常量定义的格式是由常量名与常量值所组成。





定义常量，可以双击工作夹中程序面板中的常量表一项，在设计区会出现常量数据表，鼠标点击常量数据表表头任意地方回车即可出现空白常量行，也可在设计区通过鼠标右键选择“新常量”（Ctrl+N）方式建立，如图2-24所示。

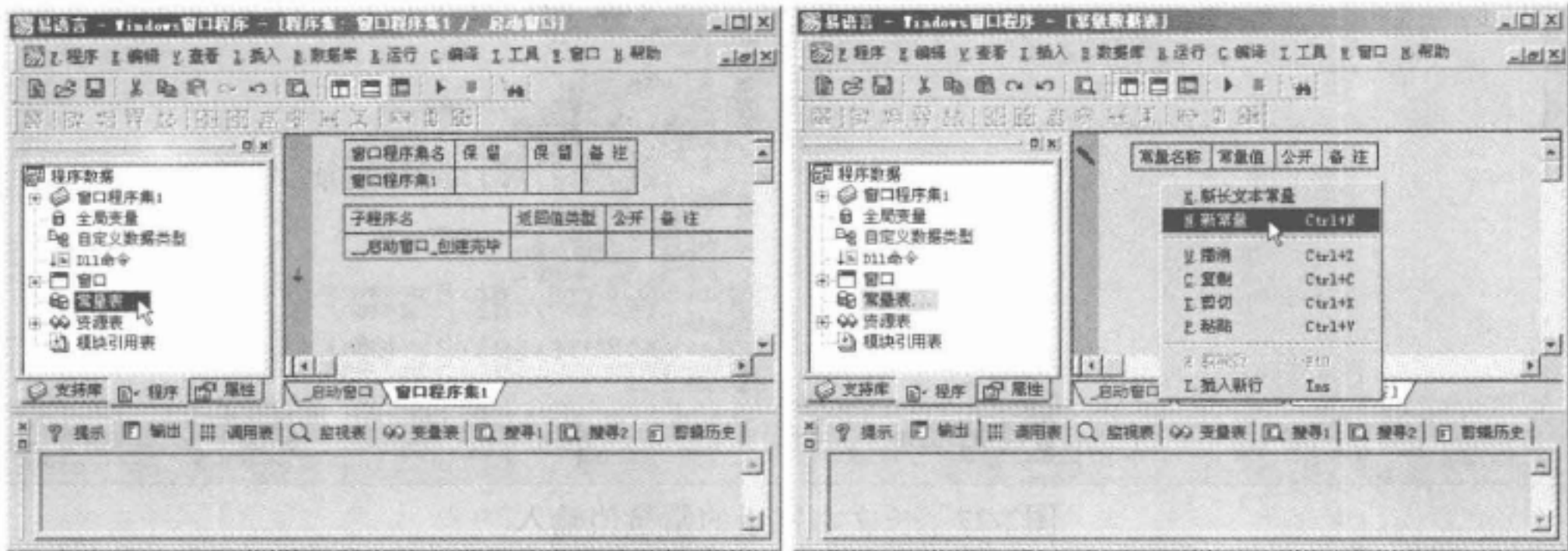


图2-24 定义常量

在当前行分别输入常量名称、常量值、备注信息（可省略），即可定义常量，如图2-25所示。

常量名称	常量值	公开	备注
开	真		逻辑型
关	假		逻辑型
WM_CLOSE	16		数值型
WM_QUIT	18		数值型
标题	“我们一起学习易语言”		文本型
中华人民共和国成立	[1949年10月1日]		日期型

图2-25 定义常量示例

1) 定义长文本常量

上图中定义了逻辑型，数值型，文本型，日期型的常量。

易语言从4.03版开始，新增了对长文本常量的支持。可以在设计区通过鼠标右键选择“新长文本常量”方式建立，如图2-26所示。

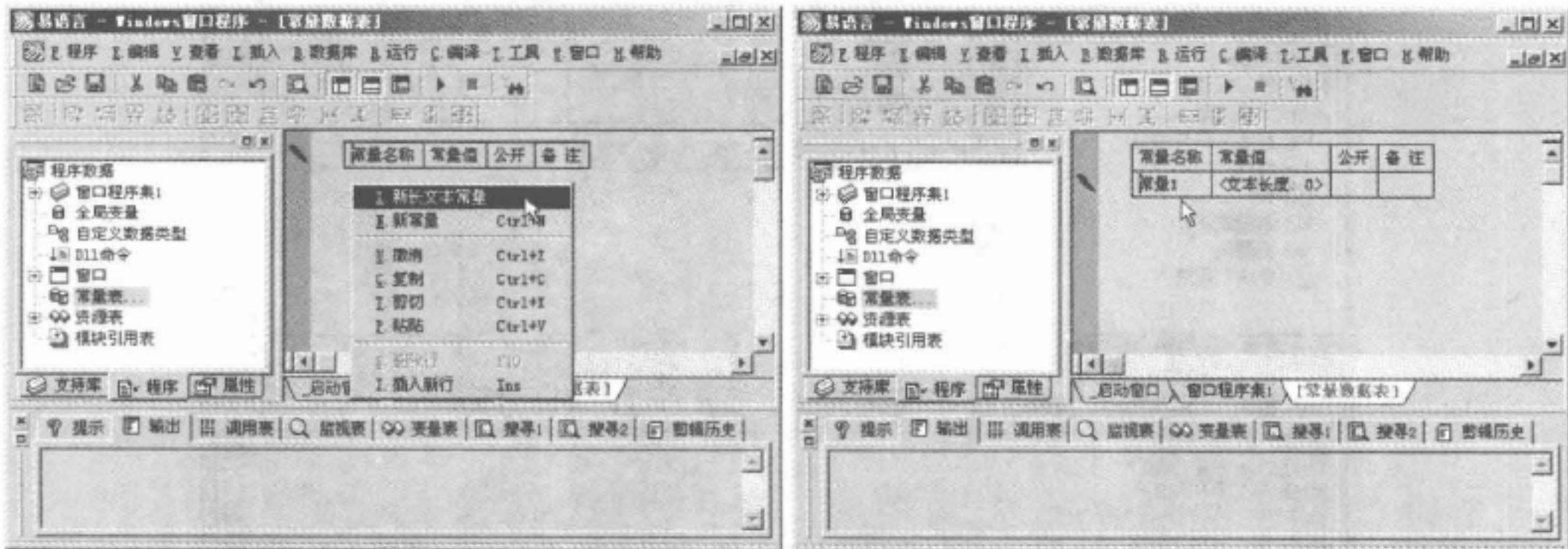


图2-26 定义长文本常量





修改常量的名称为需要设定的常量名，然后单击“<文本长度: 0>”处，弹出“请输入文本”对话框，在这里可以输入或导入需要的文本，如图2-27所示。

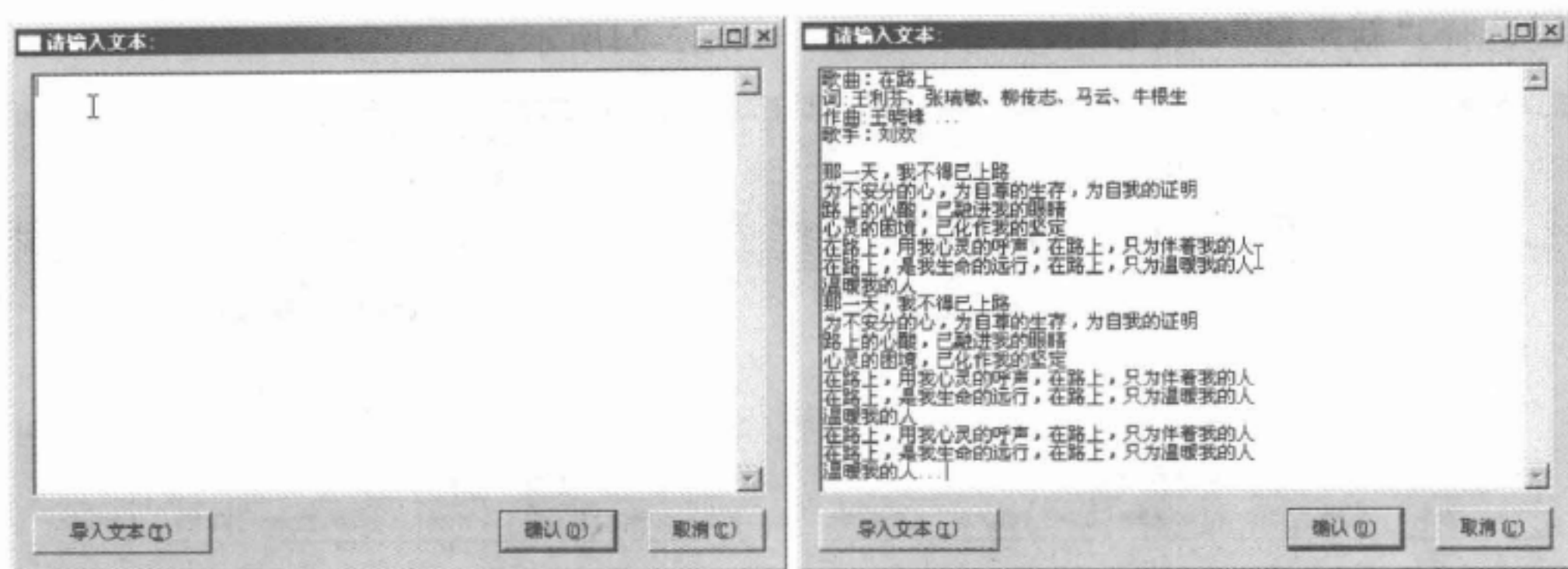


图2-27 长文本常量的常量值输入

从以上可以看出长文本方便、直观，不但保存换行的效果，而且方便实现多行文本的常量保存。点击确定后可以看到该常量的文本长度，如图2-28所示。

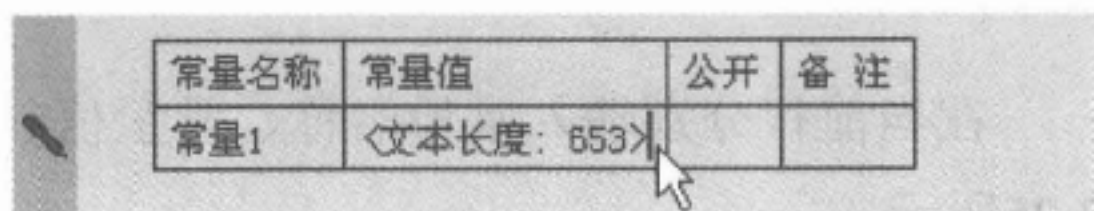


图2-28 长文本常量的长度

长文本常量和其他常量的调用方式相同，只是存储容量更大，最大支持约30000字节。

## 2) 常量的调用方式

常量在代码里使用方式为：#常量名。如：在程序启动后要显示系统的标题。很多时候可以将标题定义为常量，便于程序的修改和访问。

定义一个名称为“标题”的常量，设置常量值为“我们一起学习易语言”。

双击启动窗口进入代码编辑界面，光标处输入以下代码：

```
_启动窗口.标题 = #标题
```

按F5键运行程序，可以看到窗口的标题为“我们一起学习易语言”，如图2-29所示。

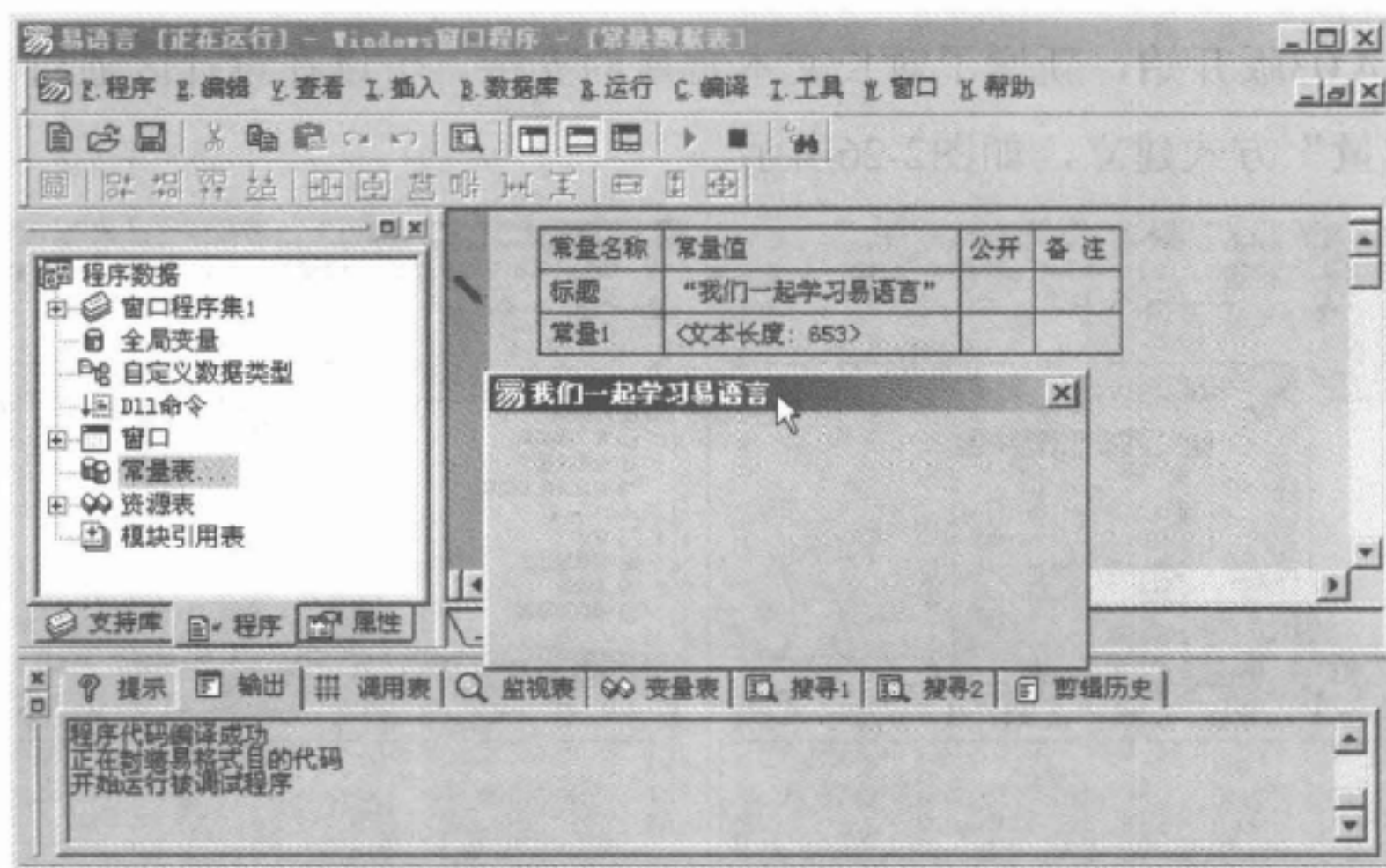


图2-29 调用常量





### 3) 支持库常量

除了以上介绍的用户自己定义的常量外，易语言的核心支持库、扩展支持库也定义了许多常量，这些常量可以直接用“#常量名”调用。

可以在易语言的支持库面板对各支持库中的定义的常量进行查询。展开一个支持库，看看该支持库是否提供了“常量”，如果在被展开项目中找到了“常量”选项，点击“常量”，就可以展开该支持库中的常量列表。点击一个常量后，按下“F1”键，可以在底部的提示框中看到有关该常量的帮助信息，并可以查到该常量的值，如图2-30所示。

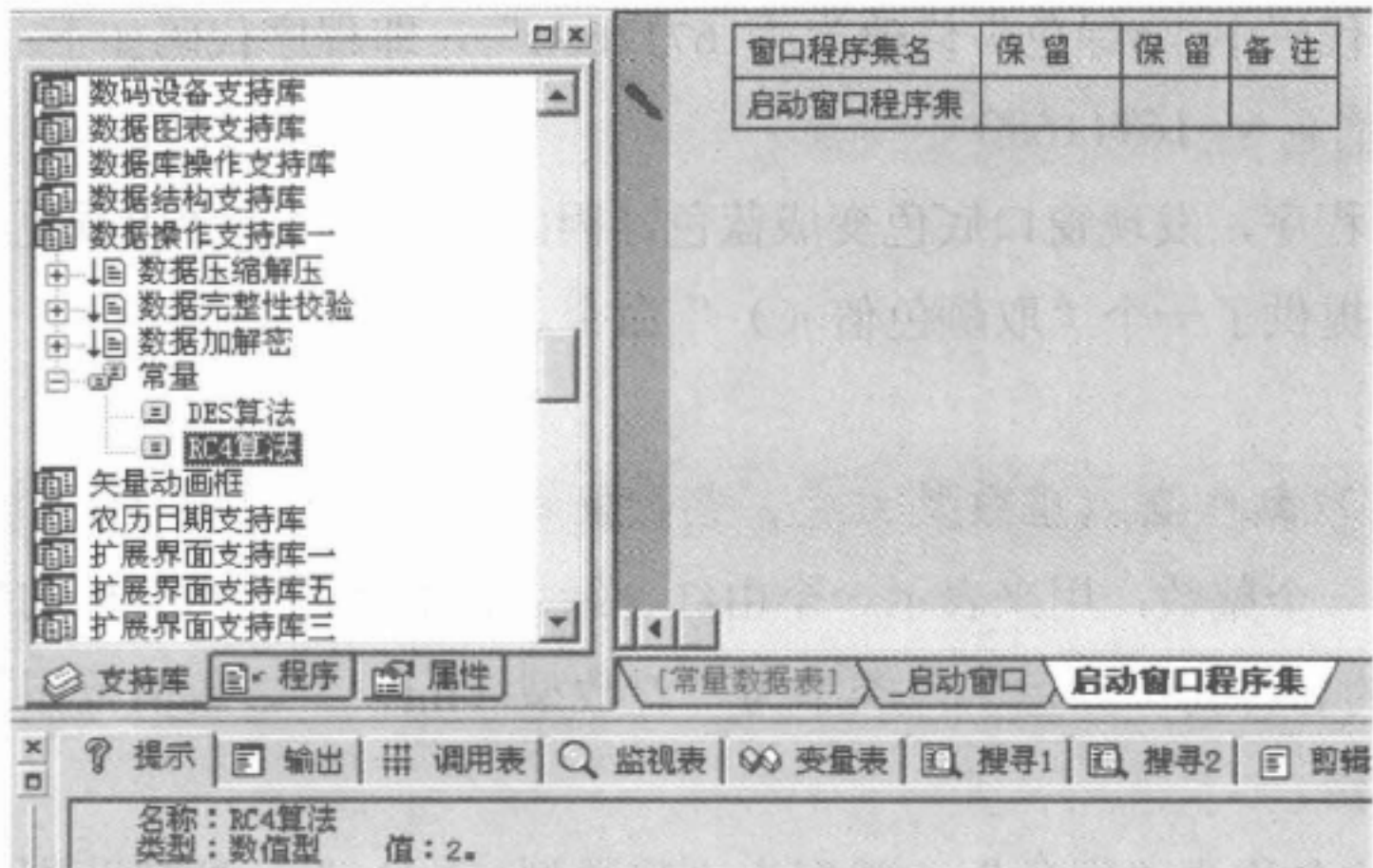


图2-30 支持库中的常量查看

每次安装了新的支持库，也可以使用该方法来查看新支持库中引用了哪些常量，以方便程序的编写。系统核心支持库提供的常量是使用频率最高的常量，应该重点掌握。

(1) 颜色常量的调用：易语言提供了33个颜色常量，这些常量可以在系统核心支持库的常量部分查到。

有颜色属性的组件，在颜色属性上都有一个颜色选择器，用来直接改变颜色。在颜色选择器上可直接选择的颜色，其值都可以作为常量，在调用的时候只需要输入“#颜色名”即可。例如：双击启动窗口进入代码编辑界面，光标处输入以下代码：

```
_启动窗口.底色 = #绿色
```

按F5键运行程序，显示一个底色为绿色的窗口。

在编程中，每一种颜色都有一个颜色值，这个颜色值是怎么来的呢？

在电脑中为了方便区分和应用颜色，用二进制16位、24位、32位、64位计算，把每一种颜色按颜色渐变顺序排列，即形成了颜色代码。

颜色代码可以用RGB（红绿蓝）值或者十六进制代码（hex）值来表示，比如红色RGB值为255，0，0；换成十六进制为#ff0000。

在RGB中从0~255的三个值中，0是最低阶的（如没有红色），255是最高阶（如全是红色），也就是该颜色的饱和度。





这些值也可以是百分比。如RGB (255, 0, 0) 可以用 RGB (100%, 0%, 0%) 表示。

十六进制有16个数字，从0至F (0~9是数字，10~15分别用ABCDEF表示)。

在颜色中十六进制用#号后面跟随3位或6位数字表示。如红色#ff0000可以表示成#f00。

但用6位表示可以更精确表示色彩。

在易语言中颜色值是一个整数型数据，它的内容是一个十进制数值，可以表示32位颜色值。如蓝色RGB (0, 0, 255)，十进制为：16711680，十六进制为：#0000FF。

将上面例程代码中“#绿色”修改为“16711680”，即程序代码如下：

```
_启动窗口.底色 = 16711680
```

按F5键运行程序，发现窗口底色变成蓝色，因此也可以直接使用颜色值来赋值。

在易语言中提供了一个“取颜色值 ()”命令。

命令原型为：

<整数型> 取颜色值 (整数型 红色, 整数型 绿色, 整数型 蓝色)

该命令返回一个整数，用来表示一个由红、绿、蓝的饱和度组合成的颜色值。

参数<1>的名称为“红色”，类型为“整数型 (int)”。数值范围从0~255，表示颜色中的红色成份。

参数<2>的名称为“绿色”，类型为“整数型 (int)”。数值范围从0~255，表示颜色中的绿色成份。

参数<3>的名称为“蓝色”，类型为“整数型 (int)”。数值范围从0~255，表示颜色中的蓝色成份。

假设知道粉红色的RGB颜色值为 (255, 192, 203)，以上的程序代码修改如下：

```
_启动窗口.底色 = 取颜色值 (255, 192, 203)
```

按F5键运行程序，发现窗口底色变成粉红色。

(2) “#换行符”的调用：“#换行符”常量等同于文本 <回车> + <换行>。

如果想让编辑框显示一段文本并自动换行，就需要使用换行符，否则文本会一直排下去。如要分行显示一首唐诗，添加一个按钮和编辑框，在按钮的单击事件里输入代码，不使用换行符时的代码与使用换行符时的程序代码分别如图2-31所示。

子程序名	返回值类型	公开	备注
_按钮1_被单击			
编辑框1.内容 = 编辑框1.内容 + “白日依山尽，”			
编辑框1.内容 = 编辑框1.内容 + “黄河入海流。”			
编辑框1.内容 = 编辑框1.内容 + “欲穷千里目，”			
编辑框1.内容 = 编辑框1.内容 + “更上一层楼。”			

子程序名	返回值类型	公开	备注
_按钮1_被单击			
编辑框1.内容 = 编辑框1.内容 + “白日依山尽，” + #换行符			
编辑框1.内容 = 编辑框1.内容 + “黄河入海流。” + #换行符			
编辑框1.内容 = 编辑框1.内容 + “欲穷千里目，” + #换行符			
编辑框1.内容 = 编辑框1.内容 + “更上一层楼。” + #换行符			

图2-31 使用换行符时代码比较

按F5键运行程序，显示效果分别如图2-32所示。





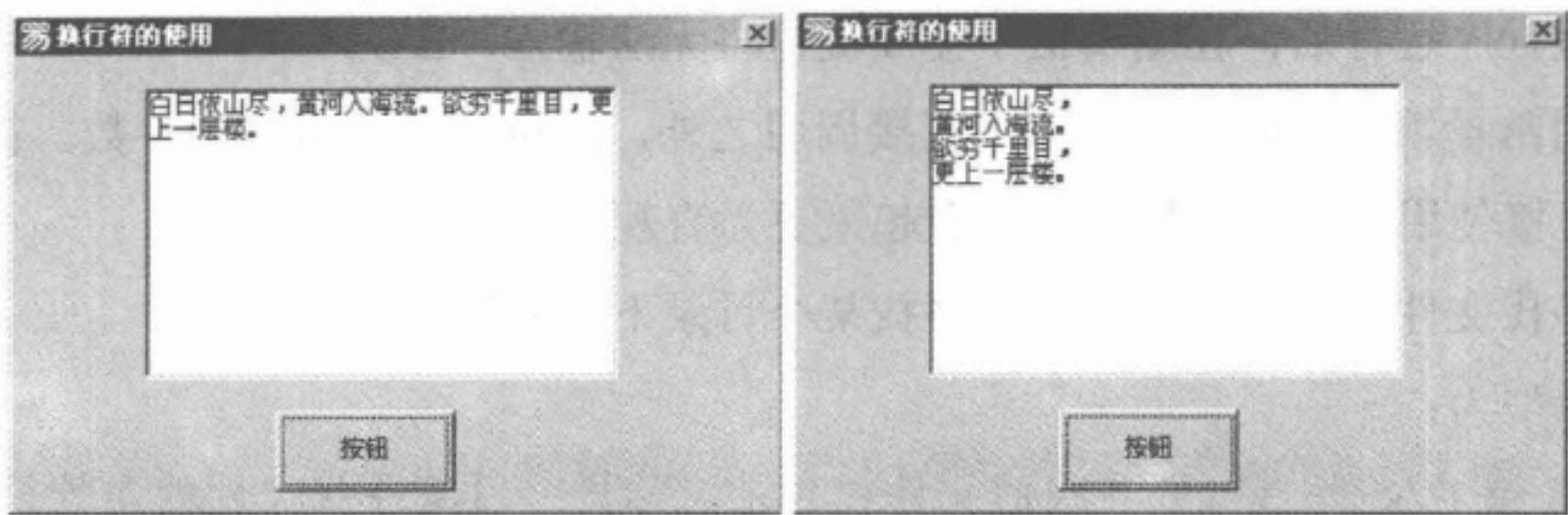


图2-32 使用换行符时效果比较

**注解：**显示多行的基本条件是将编辑框的“是否允许多行”属性设为真。可以直接修改编辑框的“是否允许多行”属性，或在代码中添加：

编辑框1.是否允许多行 = 真

(3) “#引号”、“#左引号”、“#右引号”：为了不和代码中表示文本数据的引号相冲突，程序代码中将文本的引号作为了一个文本常量。如让编辑框显示出一个引号，要使用“#引号”常量，但要显示汉语标点中的引号，就要使用常量“#左引号”、“#右引号”。

例如：新建一个易程序，添加2个编辑框，双击启动窗口进入代码编辑界面，输入如下代码：

编辑框1.内容 = #左引号 + “我爱易语言！” + #右引号

编辑框2.内容 = #引号 + 大家一起学！ + #引号

按F5键运行程序，显示效果如图2-33所示。

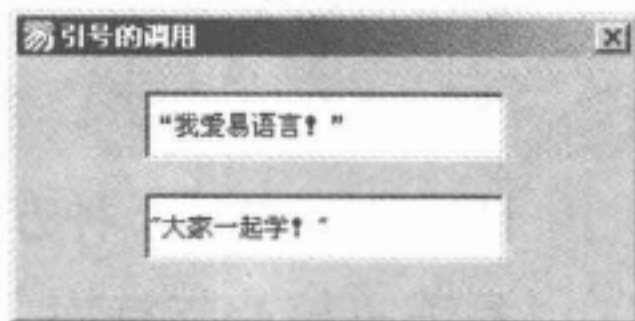


图2-33 引号常量的调用

(4) 键代码常量的调用：易语言中，将键盘上常用按键的键代码都作为了核心支持库定义的常量，在系统核心支持库的常量位置处可以找到，如图2-34所示。

在程序中使用时只需输入“#”+要调用的键名，如键盘上F8的键代码，在代码中用常量表示为：#+F8键。

例如：在向编辑框中输入内容的时候，想简单地屏蔽掉某个键，就可以在编辑框的“按下某键”事件子程序中输入以下代码：

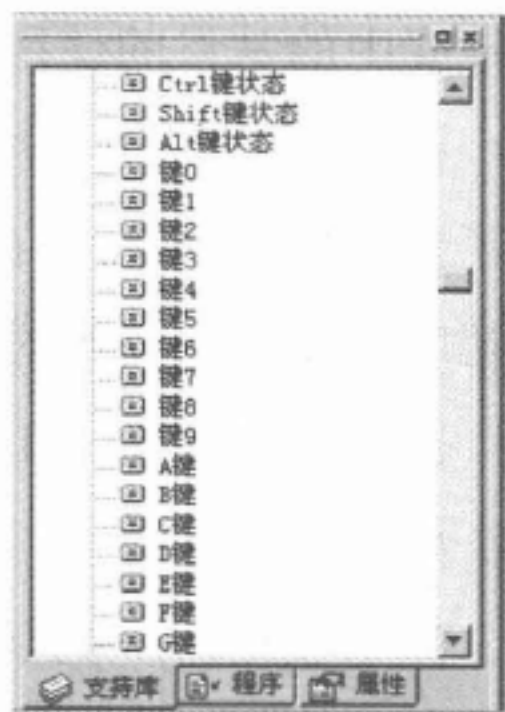


图2-34 键代码常量查看

子程序名	返回值类型	公开	备注		
_编辑框1_按下某键	逻辑型				
参数名	类型	参考	可空	数组	备注
键代码	整型				
功能键状态	整型				

--- 如果真 (键代码 = #E键)  
返回 (假)





按F5键程序运行后，在编辑框中使用键盘将无法输入“E”。

(5) 用常量填写参数。常量除了直接调用之外，还可以作为命令的参数。很多命令的参数可以直接使用常量，这样使程序看起来更为直观。

比如寻找文件（）命令。它可以寻找某个目录下的文件或目录。

命令原型为：

文本型 寻找文件（[欲寻找的文件或目录名称]，[欲寻找文件的属性]）

它的第二个参数是：“欲寻找文件的属性”，表示要寻找何种类型的文件：1（#只读文件）；2（#隐藏文件）；4（#系统文件）；16（#子目录）；32（#存档文件）。通过这些常量值加起来可以一次设置多个文件属性。如果省略本参数，默认为搜寻除子目录外的所有文件。

在寻找目录时可以直接输入：

寻找结果 = 寻找文件（, 16）

也可以输入：

寻找结果 = 寻找文件（, #子目录）

比较而言，后一种方法更直观且便于理解。

易语言核心支持库之外的支持库提供的常量称为扩展支持库常量。扩展支持库中提供的常量的使用方法与核心支持库中的常量的使用方法相同。

在易语言编辑环境的菜单—工具—支持库配置项下添加了新的支持库后，在易语言的支持库面板就会出现相应的帮助信息。在使用时可以查询支持库的常量。

比如添加了mysql支持库之后，可以看到该支持库的常量，如图2-35所示。



图2-35 mysql支持库常量查看

如果使用mysql支持库的“到MYSQL文本（）”命令。该命令的作用是从易语言中的数据类型转换到可以被MYSQL识别的文本，转换成功返回MYSQL可以识别的文本，失败返回空文本。命令原型为：

<文本型> 到MYSQL文本（易语言类型数据，时间类型）

它的第二个参数“时间类型”是要转换到MYSQL的时间与日期类的类型常量，如#MYSQL日期型，#MYSQL时间型，#MYSQL日期与时间型，#MYSQL年份型。

在编写代码时可以直接输入：

到MYSQL文本（取现行时间（），10）

也可以输入：

到MYSQL文本（取现行时间（），#MYSQL日期型）

比较而言，后一种方法更直观，便于理解。





4) 枚举常量

枚举常量是一种非常方便的常量类型，它本身就是一个常量的集合，将多个常量以成员的形式，存放在一个常量中，使用的格式为：

#枚举常量名. 成员名

枚举常量只是一种常量的表现形式，是由易语言支持库定义的常量集合，调用方法和普通常量相同，用户只能调用，不能自定义。

易语言中有很多支持库中使用了枚举常量，如核心支持库中定义的“变体类型”。“变体类型”提供变体型中所能够容纳数据类型的枚举值，见表2-3。

表2-3 变体类型常量成员及常量值

成员	描述	成员	描述
未知	常量值为 -1	数值型数组	常量值为 7
空	常量值为 0	文本型数组	常量值为 8
数值型	常量值为 1	逻辑型数组	常量值为 9
文本型	常量值为 2	日期型数组	常量值为 10
逻辑型	常量值为 3	对象型数组	常量值为 11
日期型	常量值为 4	错误值型数组	常量值为 12
对象型	常量值为 5	变体型数组	常量值为 13
错误值型	常量值为 6		

表2-3中，列出了“变体类型”常量的所有成员名及成员的常量值，在程序中，如果想调用“变体类型”常量中的“日期型”成员，该成员的常量值为4，程序中调用该成员就等于调用了4这个整数，例如用信息框显示出该成员使用代码：

信息框(#变体类型.日期型,0,)

运行后，信息框将显示4。

**注解：**在核心支持库中还定义了“变体型”。“变体型”和“变体类型”不同，“变体型”是一种数据类型，可以将一个变量的类型定义成“变体型”。“变体型”的变量，可以加入成员和改变成员的值；而“变体类型”是一个枚举常量，其值只可以调用而不可以改变。

再如办公组件支持库中定义的“对象种类”，“对象种类”同样提供了一组枚举值，见表2-4。

表2-4 对象种类常量成员及常量值

成员	描述	成员	描述
圆	常量值为 1	折线	常量值为 5
圆形印章	常量值为 2	十字多边形	常量值为 6
圆锥	常量值为 3	立方体	常量值为 7
曲线	常量值为 4	圆柱	常量值为 8



续表

成员	描述	成员	描述
椭圆	常量值为 9	方形印章	常量值为 18
四角星	常量值为 10	旋转字	常量值为 19
图片	常量值为 11	圆角矩形	常量值为 20
直线	常量值为 12	表格	常量值为 21
五角星	常量值为 13	文字圆	常量值为 22
多边形	常量值为 14	文本框	常量值为 23
正多边形	常量值为 15	水印	常量值为 24
棱锥	常量值为 16	图表	常量值为 25
矩形	常量值为 17	OLE对象	常量值为 26

使用“#枚举常量名. 成员名”的形式调用。如：

子程序名	返回值类型	公开	备注
_表格_被选择			

对象接口. 插入 (#对象种类. 表格)

可以看出枚举常量和和其他常量的用法都是一样的，只不过一般常量用“#常量名”，而枚举常量所使用的是“#枚举常量名. 成员名”的形式。

## 2.1.4 资源表

在编程的时候，常会使用图片和声音文件，在程序发布的时候需要将使用到的图片、声音文件一起提供，软件才能正常使用，因此极为不方便。但在易语言中提供资源表，这使得程序的开发与使用更方便、快捷，如图2-36所示。

这里可以存放程序开发中用到的图片、声音等二进制数据资源。这样可以方便的调用这些资源，由于这些文件保存在程序内部，因此相对来说保护这些原文件不被修改和复制。值得注意的是添加后的资源都是字节集型的数据。

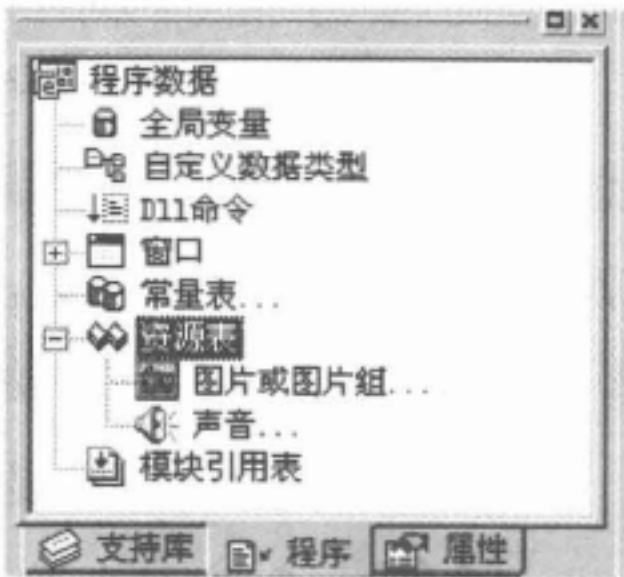


图2-36 资源表位置

### 2.1.4.1 添加图片或图片组资源

添加图片或图片组资源的方法和新建常量的方式基本类似。双击工作夹中程序面板的资源表中的“图片或图片组...”选项，即可切换到添加图片资源的界面，直接回车或鼠标右键选择“新图片或图片组资源”（Ctrl+N），即可建立，如图2-37所示。

在新建的“图片1”资源对应的内容上单击鼠标左键或按下键盘中的任意字符键，会弹出一个“图片或图片组资源属性”对话框，如图2-38所示。可以点击对话框中的“导入新图片”按钮或“图片组...”按钮来给新的资源加入图片或图片组。

加入需要的图片后，点击“加入 / 修改到程序并关闭对话框（A）”按钮，保存添加、修改的内容，即可完成整个图片资源的添加、修改过程。



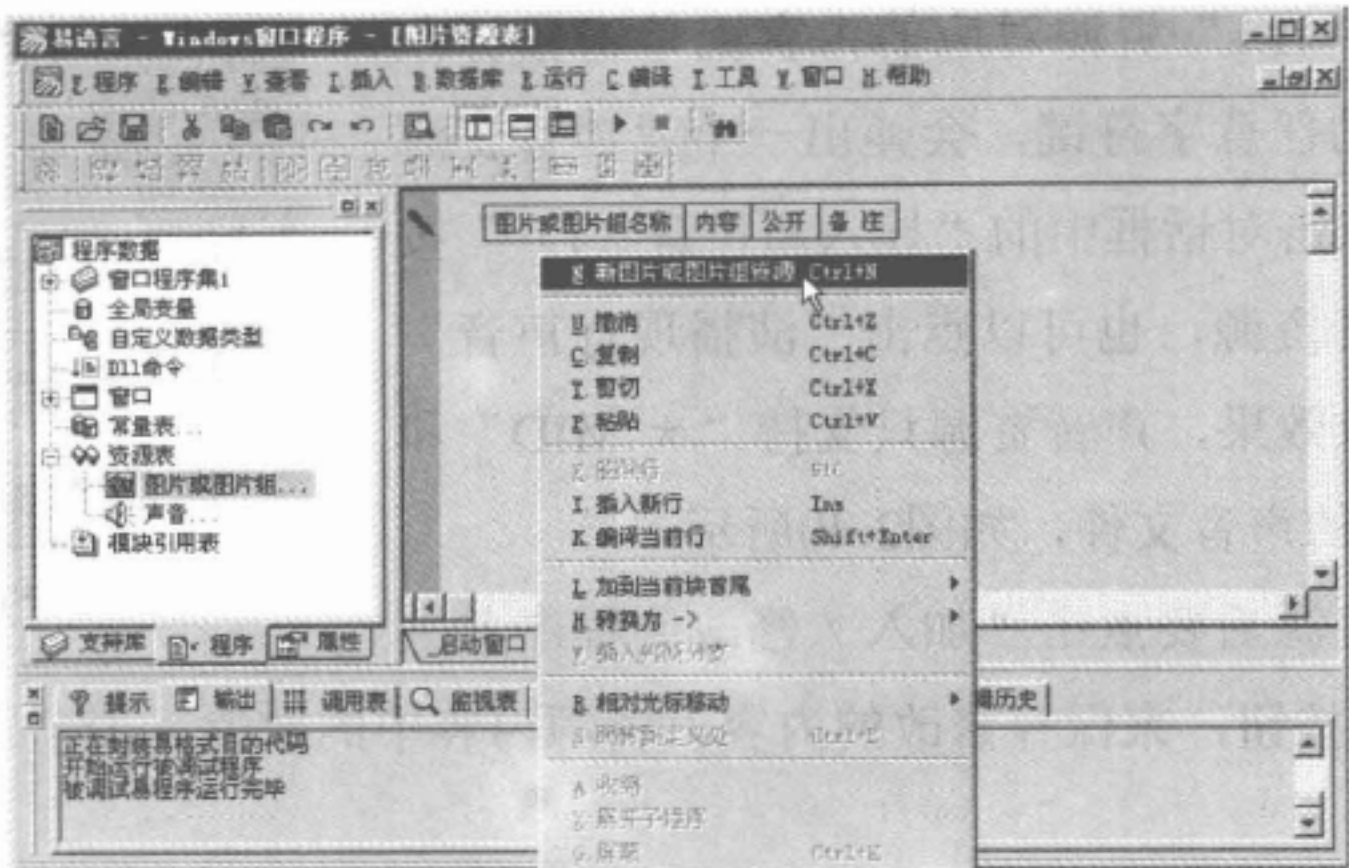


图2-37 添加图片资源

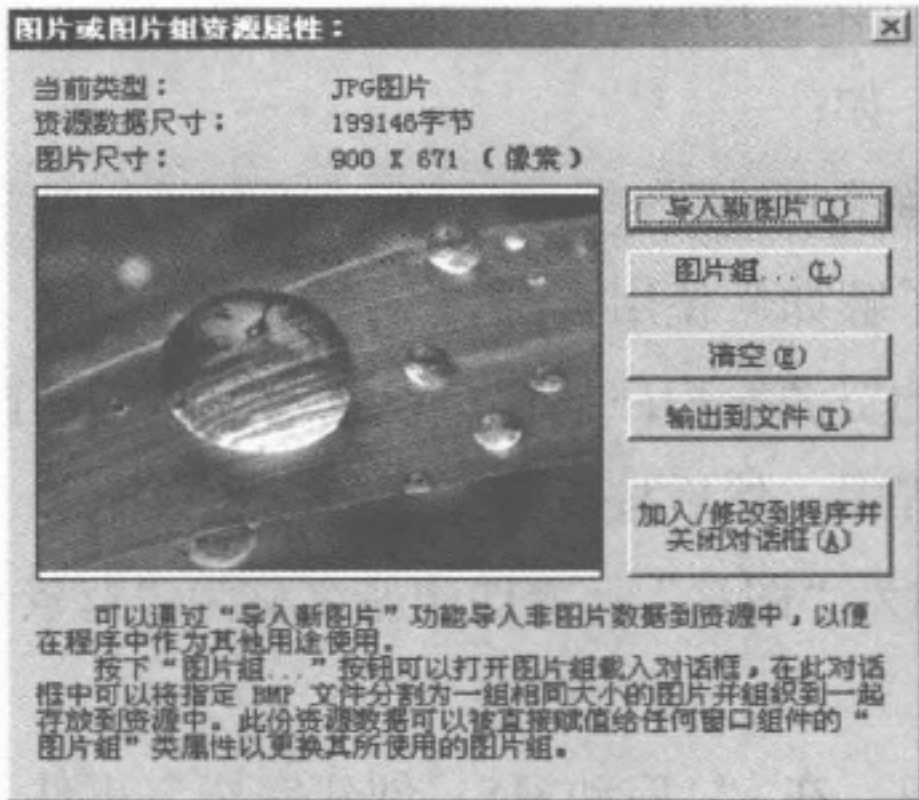


图2-38 图片或图片组资源属性对话框

2.1.4.2 添加声音资源

添加声音资源与添加图片或图片组资源的操作基本类似，双击资源表中的“声音”选项，切换到添加声音资源的界面，直接回车或鼠标右键选择“新声音资源”（Ctrl+N），即可建立，如图2-39所示。

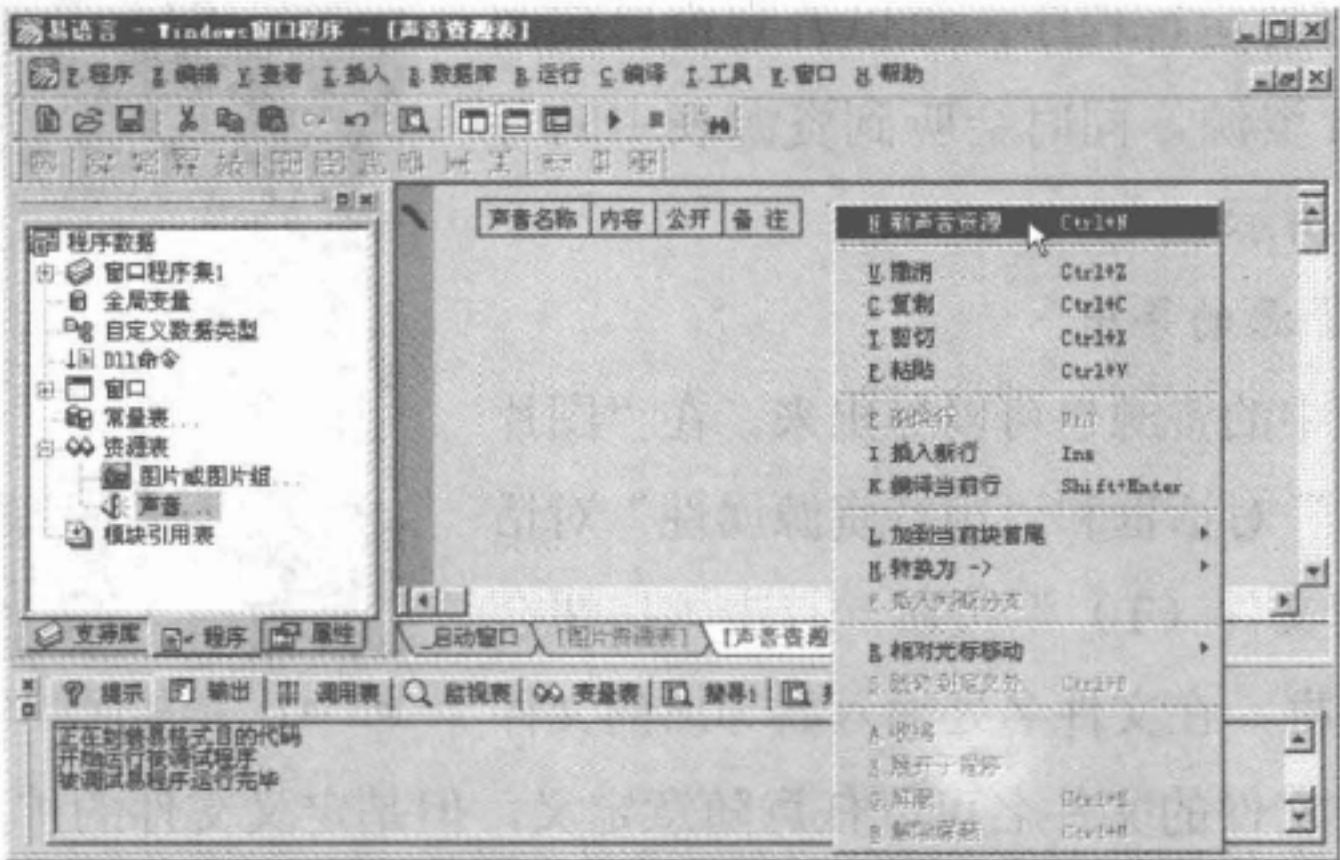


图2-39 添加声音资源





在新建的“声音1”资源对应的内容上单击鼠标左键或按下键盘中的任意字符键，会弹出一个“声音资源属性”对话框。点击对话框中的“导入新声音(I)”按钮来加入新的声音资源，也可以点击“演播现行声音”按钮直接试听播放效果，声音资源只支持“\*.MID”和“\*.WAV”格式的声音文件，如图2-40所示。

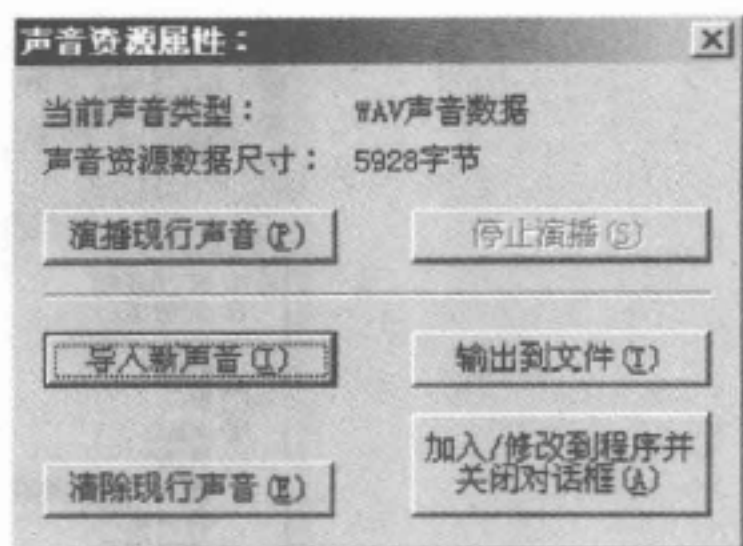


图2-40 声音资源属性对话框

加入了声音资源后要点击“加入/修改到程序并关闭对话框(A)”按钮，来保存修改的内容。资源内容中的数值，表示了资源所占用的字节数。

### 1) 资源的调用

添加后的资源都是字节集型的数据，在程序中调用资源的方法和常量的调用方法相同，用“#”+“资源名”。如：

图片框\_资源. 图片=#图片资源

下面通过一个例程来了解如何使用资源。

**【例】** 调用资源表中的资源给程序的窗口设置底图和背景音乐。

第一步：新建一个易程序，然后按前面介绍的方法新加入一个图片资源和一个声音资源，修改图片或图片组名称为“图片”；修改声音名称为“背景音乐”，并分别加入图片和背景音乐。

第二步：双击启动窗口，在“\_启动窗口\_创建完毕”事件子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

\_启动窗口.底图 = #图片

\_启动窗口.底图方式 = 3

\_启动窗口.背景音乐 = #背景音乐

第三步：按F5键运行程序，可以看到窗口底图变成了添加的图片资源，同时会听到资源表中的背景音乐，如图2-41所示。

### 2) 资源表中资源的导出

存放在资源表中的资源也可以导出来。在“图片或图片组资源属性”对话框和“声音资源属性”对话框中都有“输出到文件(T)”按钮，点击该按钮会弹出一个文件对话框，在文件名处输入欲导出的文件名和扩展名。其中文件的文件名可以任意随意定义，但是定义文件的扩展名一定要和原文件的扩展名相同，否则保存后的文件有可能无法正常打开。也可以使用“写到文件(I)”

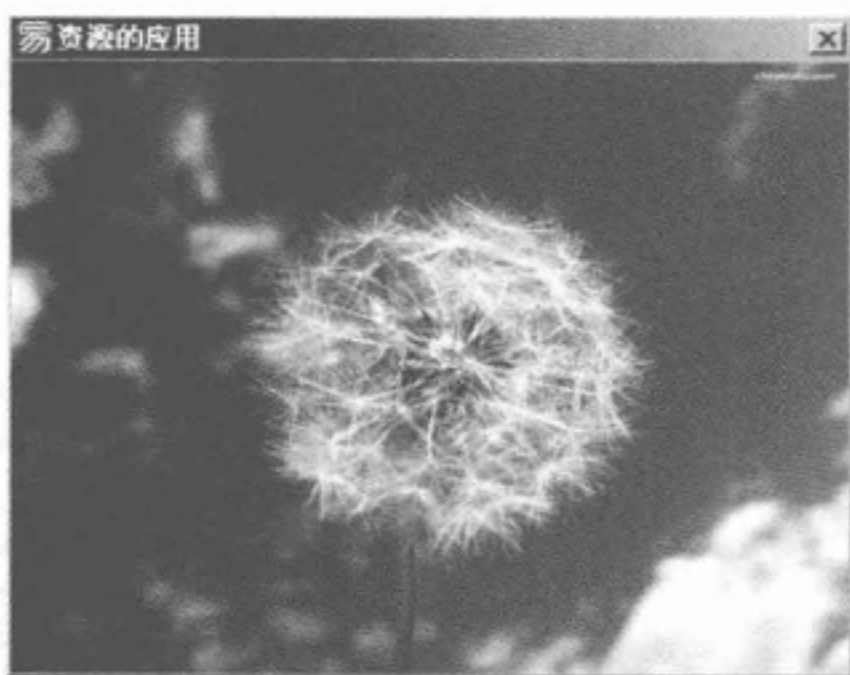


图2-41 添加背景图片和背景音乐的窗口





命令导出资源，但需要注意文件的扩展名是否与源文件相同。

如：导出资源表中的图片资源“背景图片”到“D:\tupian.bmp”。代码如下：

写到文件（“D:\tupian.bmp”，#背景图片）

按F5键运行后，将资源表中的图片资源“背景图片”写到D盘根目录中的“tupian.bmp”文件中。

### 3) 资源表中导入可执行文件

资源表中不仅可以导入图片和声音文件，还可以导入任意文件，程序不会判断该资源是不是一个图片或声音。如：导入一个文本文件；一个word文件或者一个“.exe”格式的可执行文件等。

例如：导入windows中自带的记事本程序，在程序运行时导出并运行。

以图片资源表为例：在图片资源表中加入一个新资源，然后点击弹出的“图片或图片组资源属性”对话框中的“导入新图片”按钮，然后将弹出的通用对话框文件类型改为所有文件 (\*.\*)，找到欲导入资源表的windows记事本“notepad.exe”文件，如图2-42所示。



图2-42 导入可执行文件

然后点击“打开”按钮，将该可执行文件导入到资源表，导入之后点击“加入 / 修改到程序并关闭对话框”按钮，保存。

用“写到文件（）”命令写出，并用“运行（）”命令运行。代码如下：

写到文件（“D:\记事本.exe”，#记事本）

运行（“D:\记事本.exe”，假，）

按F5键运行程序，会将资源表中的记事本程序写出到D盘的根目录下面，并运行。

## 2.2 运算符和表达式

运算符是作用于数据上的符号，代表了作用于数据上的特定操作。

程序中经常会使用运算符进行识别处理，并通过运算表达式来完成运算操作。程序中





对各数据之间的关系的描述也要通过运算符。所以编写代码时，除了大量地使用命令或对组件的属性、命令进行操作，运算符的使用也非常重要。

## 2.2.1 运算符

运算符根据操作的性质进行分类，可以分为：算术运算符、比较运算符、逻辑运算符和赋值运算符。

易语言中提供了大量的运算符，见表2-5。

表2-5 易语言中的运算符


运算符分类	运算符	运算符含义	代码中显示	举例	结果
算术运算符	+	加法运算，将加号两边的数相加	+	4+3	7
	-	减法运算，将减号两边的数相减；负号	-	21-5、-5	16、-5
	*	乘法运算，将乘号两边的数相乘	×	4×5	20
	/	除法运算，将除号两边的数相除	÷	30 / 6	5
	\	整除运算，将整除号两边的数整除	\	21\4	5
	%	求余数运算	%	1234%100	34
关系运算符	>	判断是否大于	>	3>10	假
	<	判断是否小于	<	5<9	真
	= 或 ==	判断是否等于	=	9=7	假
	>=	判断是否大于等于	≥	32>=12	真
	<=	判断是否小于等于	≤	23<=12	假
	<> 或 !=	判断是否不等于	≠	2<>3	真
	? =	判断是否约等于	≈	“ab”? = “a”	真
逻辑运算符	&& 或 qie	逻辑与运算符，可以连接几个必须同时满足的条件	且	2<6且10>11	假
	或 huo	逻辑或运算符，可以连接几个可选条件	或	7<5或23>14	真
	取反() 或 not	逻辑非运算符	取反	取反(2>3)	真
赋值运算符	=	将等号后面的值赋值给等号前面的对象	=	变量1=45	

程序中的运算符都有其优先级别，在程序运行的时候会按照符号的优先级别，从高到低依次运行，见表2-6。





表2-6 易语言常用运算符的优先级

运算符	优先级
()(小括号)	最高
*(乘)、/(除)	
\(整除)	
%(求余数)	
+(加)、-(减)	
比较运算符	
&&(逻辑与)	
(逻辑或)	
=(赋值)	

2.2.2 表达式

用运算符和括号将常量、变量、常数等对象连接起来的，符合易语言语法规则的式子，称易语言表达式。

【例】 下面是一个合法的易语言表达式：

```
变量1=((3 × 9 - 15 ÷ 3) + 2) \ 8
```

表达式中运算的先后，是按照运算符的优先级别来进行判定的。

算式计算的结果可以被程序调用。

【例】 将上面算式的结果用信息框显示出来。

```
信息框 (((3 × 9 - 15 ÷ 3) + 2) \ 8, 0, )
```

运算符的优先级可以通过括号来改变，括号中的表达式将首先被计算。

可以用括号改变优先顺序，强令表达式的某些部分优先运行。括号内的运算总是优先于括号外的运算，但是，在括号之内运算符的优先顺序不变。

当出现括号嵌套时，最内层的括号最先计算。

当表达式中出现多对括号前后排列时，按照从左到右的次序计算。

另外，易语言的编辑器会自动对表达式的优先级进行判断，当发现某部分表达式的优先级高，不用使用括号也会达到同样的效果时，会自动将括号省略掉。

【例】

```
变量1 = 9 × 3 > 10 且 (2 × 4 - 8 ÷ 4 > 6 或 (2 < 6 且 10 > 11))
```

表达式中，最后的

```
或 (2 < 6 且 10 > 11))
```

两边的括号是多余的，因为这部分表达式的外层是“或”符号，“且”符号的优先级本来就高于“或”，所以不用括号，也会优先执行这部分。易语言的编辑器会自动去掉这个括号，变成下面的语句。

```
变量1 = 9 × 3 > 10 且 (2 × 4 - 8 ÷ 4 > 6 或 2 < 6 且 10 > 11)
```



## 2.2.3 赋值运算符和赋值表达式

### 1) “=” 是赋值运算符

在程序中给变量赋值或用代码改变组件属性，大部分都是使用“=”进行赋值的，将等号后面的值赋值给等号前面的赋值对象。

#### 【例】

变量\_运算符=10

编辑框.高度=20

### 2) 赋值表达式

一个正确的赋值表达式，一定要保证欲赋的值和被赋值的对象之间的数据类型相同，不同的数据类型要转换成相同的数据类型后再赋值。

### 3) 赋值运算符“=”和比较运算符“=”的区别。

“赋值”与“等于”在易语言中的运算符都为“=”，怎样区分它们呢？

一般在易语言中的一条语句里不被小括号包裹的最左边一个“=”代表赋值。且一条语句只有一个赋值命令，其他“=”皆为逻辑比较命令“等于”。

#### 【例】

```
--- 如果真 (返回值 = 1)
    返回值 = 0
```

代码中，条件语句“如果真”中的“返回值=1”，是用比较运算符“=”进行比较，如果相等会执行下面的赋值运算“返回值=0”。

## 2.3 子程序的编写与调用

### 2.3.1 了解子程序

一个程序可能由成千上万条语句组成，为了方便、简化程序设计任务，可以将程序分割成较小的单元，这些单元被称为子程序。

子程序用来对一系列命令进行封装，实现模块化、重用及抽象，有利于程序结构化开发，使程序结构更清晰、合理。一个应用程序中的子程序，往往不必修改或只需稍作改动，便可以成为另一个程序的子程序。

在易语言中的子程序可以分为两大类：“事件子程序”和“用户自定义子程序”。



### 2.3.2 事件子程序

在前面的章节中介绍过新建程序后双击“启动窗口”、“按钮”等都可以进入代码编辑界面并会自动生成“\_启动窗口\_创建完毕”、“\_按钮1\_被单击”等相应的子程序，也可以选择组件后在属性面板中的事件列表中选择事件，生成事件子程序。这种对应组件所发生事件子程序，称作组件的“事件子程序”。可以在这些事件子程序中写入具体执行的代码。运行时，一旦这些事件产生，就会自动执行相应的子程序。

例如：添加一个按钮，在属性面板中的事件列表中选择被双击事件，就生成对应的按钮被双击事件子程序，如图2-43所示。

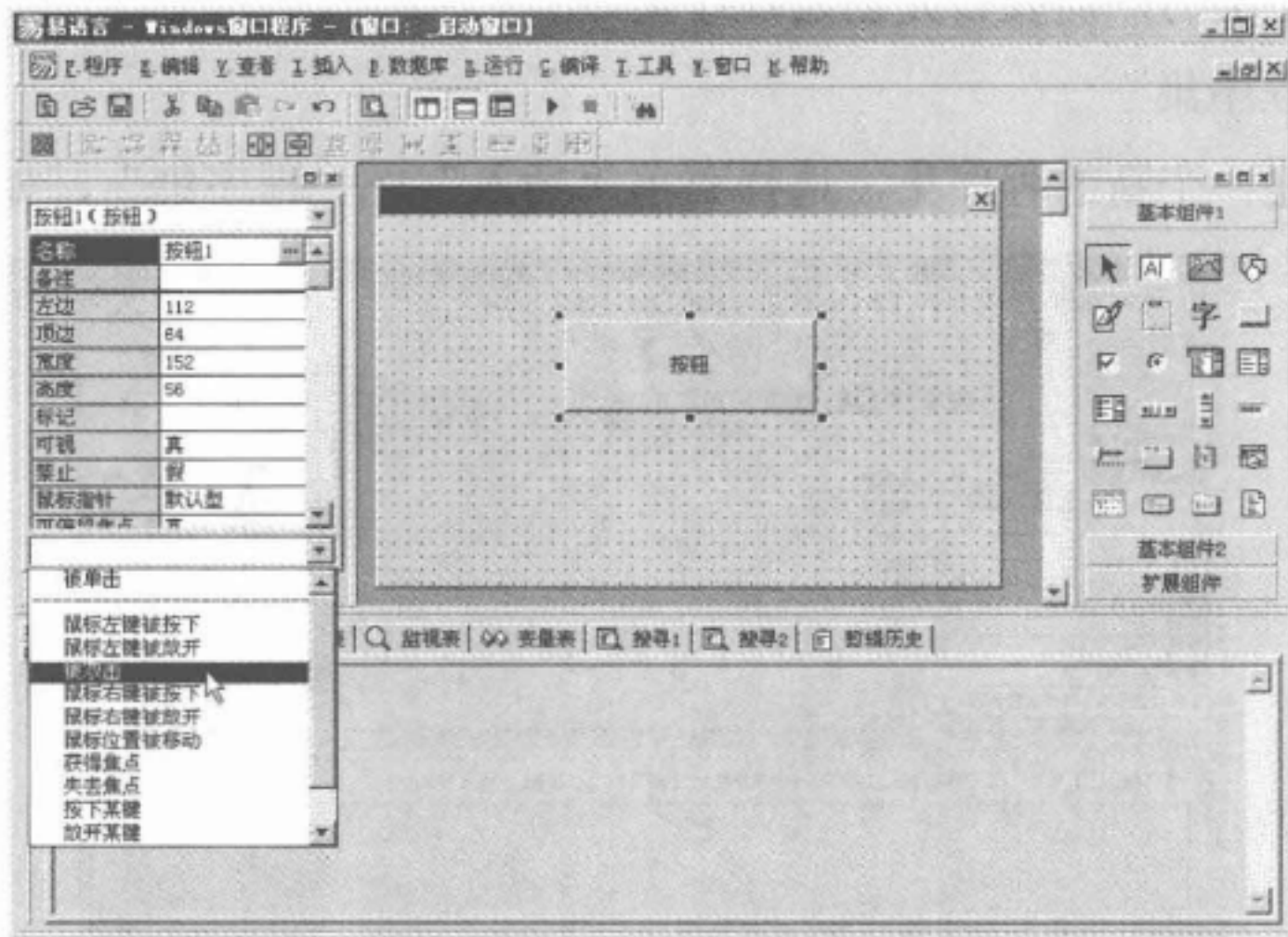


图2-43 添加按钮组件

生成“\_按钮1\_被双击”子程序，如图2-44所示。

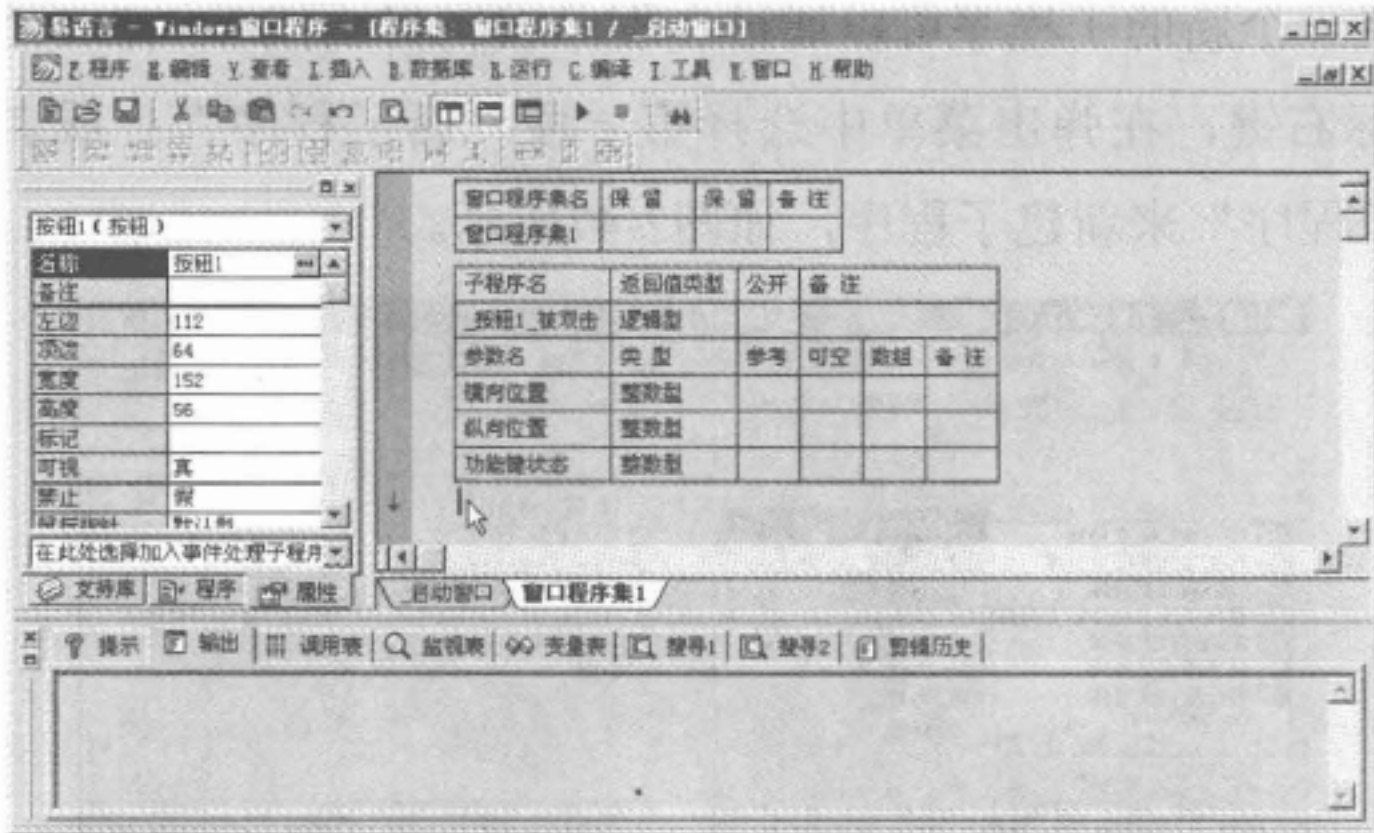


图2-44 按钮事件子程序

在“\_按钮1\_被双击”子程序下输入如下代码。

信息框 (“按钮被双击!”, 0 + #信息图标, “提示”)





按F5键运行程序，发现只有双击按钮的时候才会弹出信息框，如图2-45所示。

事件子程序的名称、返回值类型及参数个数都是系统定义的，不允许用户任意修改。

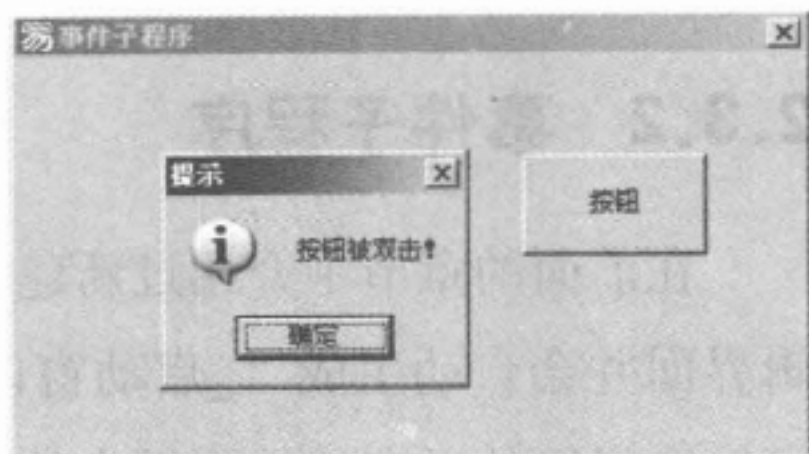


图2-45 按钮事件子程序

### 2.3.3 用户自定义子程序

“用户自定义子程序”是由用户创建，其参数个数和返回值都由用户自行定义的子程序。用户可以根据需要在程序设计时对其任意修改。

子程序是存在于程序集中的，由程序集分组管理。若一个新建的易程序还没有进行任何操作是没有程序集的，需要首先创建程序集。

#### 【例】 重建程序集

第一步：创建一个新的程序集，选择菜单“插入”→“程序集”，如图2-46所示。

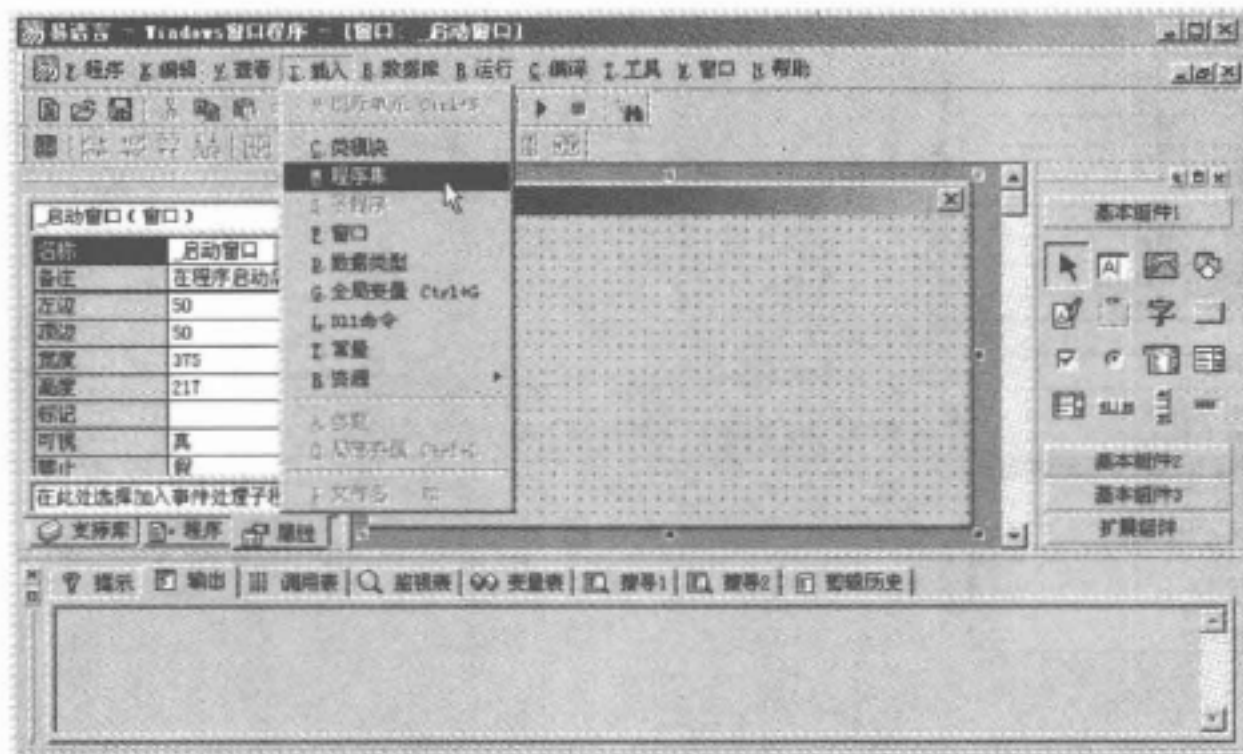


图2-46 创建一个新的程序集

有了程序集，就可以在相应的程序集中添加新的子程序了。

第二步：定义一个新的子程序可以通过在程序编辑界面按下“Ctrl+N”键；或在程序编辑界面点击鼠标右键，在弹出菜单中选择第一项“新子程序”；或者选择易语言菜单“插入”→“新子程序”来新建子程序，如图2-47所示。

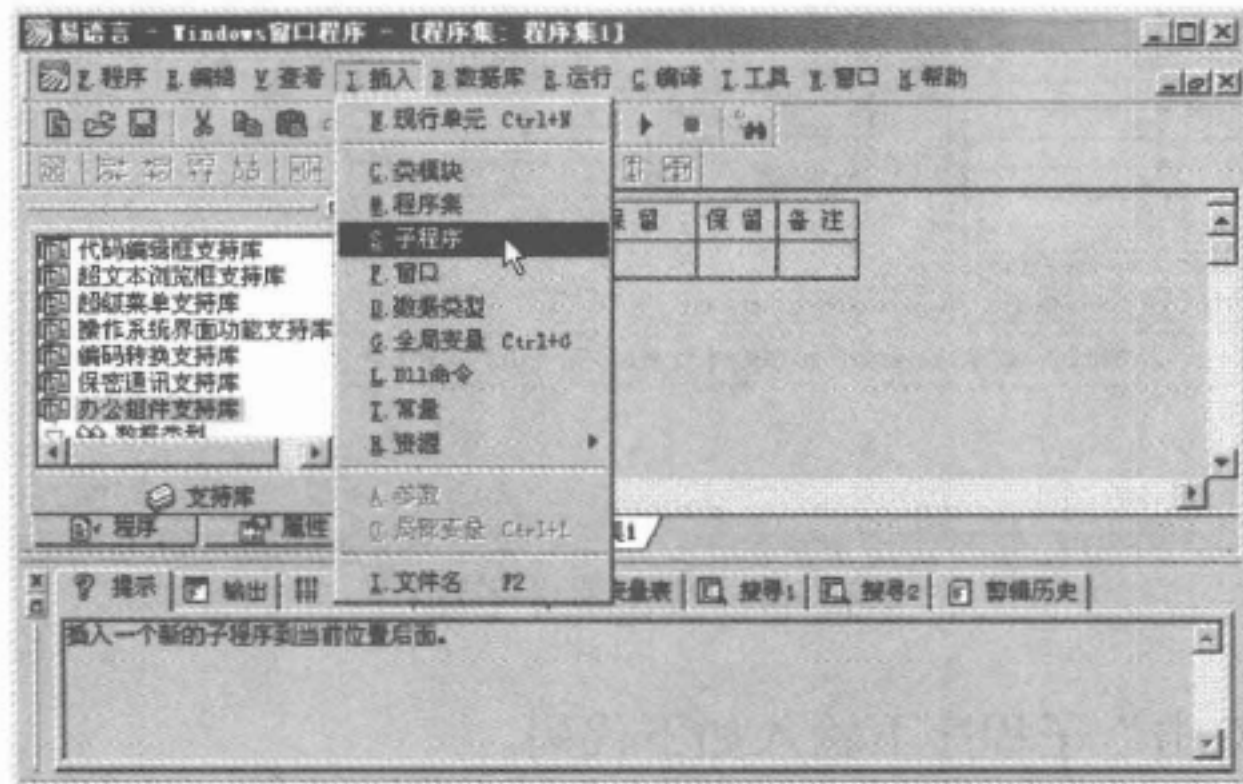


图2-47 创建一个新的程序集





生成的多个子程序默认命名方式为：子程序1，子程序2……的格式。

子程序名	返回值类型	公开	备注
子程序1			

尽量修改子程序名为能体现其功能的名称，如“添加记录”、“修改记录”等。

子程序名	返回值类型	公开	备注
添加记录			

另外也可以直接在窗体空白处双击鼠标，会自动创建该窗口对应的窗口程序集，并自动生成“\_启动窗口\_创建完毕”子程序；也可以通过双击窗口中按钮等组件，同样会创建该窗口对应的窗口程序集及组件对应缺省事件子程序，如图2-48所示。

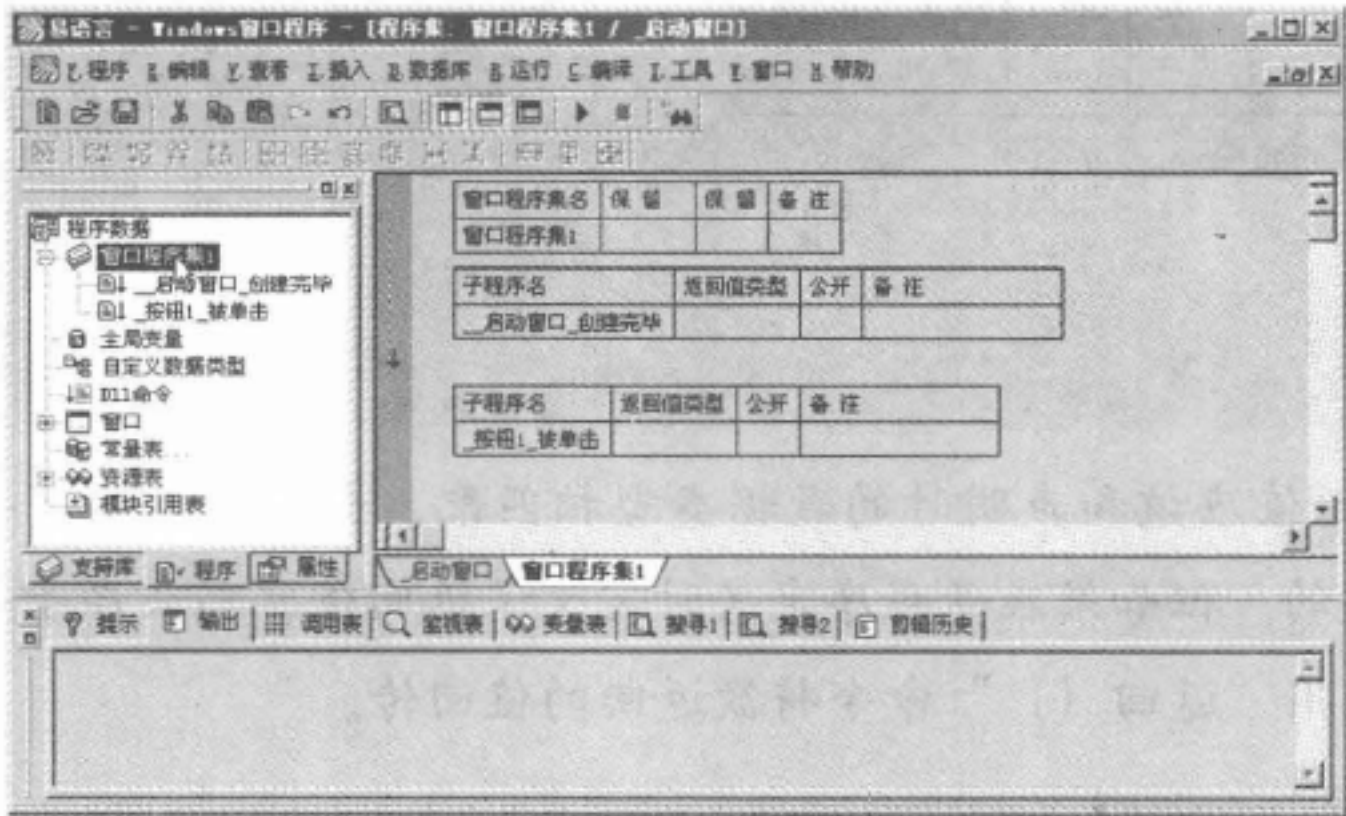


图2-48 创建窗口程序集

用户自定义子程序可以放在窗口程序集中或自定义的程序集中。不过一般为了方便引用、查找程序集变量和窗口中的组件，将当前窗口用到的用户自定义子程序都放在当前窗口程序集中。

2.3.4 子程序的返回值

子程序是一个独立完成某个功能的程序块，子程序相互之间是通过参数和返回值来联系的。如果程序代码不需要获得子程序执行后的结果值，则不必定义返回值。因此子程序可以定义返回值，也可以不定义返回值。

为子程序定义返回值，则需要子程序的“返回值类型”单元格中定义其返回值的数据类型，默认时在“返回值类型”处为空，表示无返回值。

例如：

子程序名	返回值类型	公开	备 注		
设置服务器					
参数名	类 型	参考	可空	数组	备 注
配置名	文本型				





如设置该子程序的返回值为“逻辑型”，表示需要测试配置服务器是否成功。

子程序名	返回值类型	公开	备注		
设置服务器	逻辑型				
参数名	类型	参考	可空	数组	备注
配置名	文本型				

在子程序中，通过“返回（）”命令将返回值回传给调用它的程序。一个子程序只能给调用它的程序返回一个值。

例如：

子程序名	返回值类型	公开	备注		
设置服务器	逻辑型				
参数名	类型	参考	可空	数组	备注
配置名	文本型				

’ 设置服务器的过程代码省略...

’ 如果设置成功

返回（真）

’ 如果设置失败

返回（假）

**注解：**返回的值应该和声明时的数据类型相匹配。如果“返回（-1）”代表配置服务器失败则是错误的。但如果在子程序定义时定义了返回值，那么在子程序任意一个结束分支中，都必须使用“返回（）”命令将欲返回的值回传。

### 2.3.5 子程序的调用

子程序的调用方法和命令的调用方法相同，输入子程序的名称就可以调用该子程序。

例如：定义一个无参数，无返回值的子程序“子程序\_保存数据”。

子程序名	返回值类型	公开	备注
子程序_保存数据			

调用该子程序的代码：

子程序\_保存数据（）

程序代码表示调用“子程序\_保存数据”，该子程序无参数，无返回值。

如果子程序要求提供参数，在括号中要按照子程序参数定义的数据类型和参数个数顺序填写参数。

例如：定义一个有两个参数，无返回值的子程序“子程序\_开始备份”。

子程序名	返回值类型	公开	备注		
子程序_开始备份					
参数名	类型	参考	可空	数组	备注
文件名	文本型				
文件路径	文本型				





调用该子程序的代码：

```
子程序_开始备份(“panda.ini”，“c:\save\”)
```

程序代码表示调用“子程序\_开始备份”，该子程序无返回值，有两个参数。第一个是文件名，第二个是文件路径，调用该子程序时需要提供这两个参数。

如果需要用变量保存子程序返回值，必须使用和子程序返回值相同的数据类型变量。

例如：定义一个有两个参数，有返回值的子程序“子程序\_求和”。

子程序名	返回值类型	公开	备注		
子程序_求和	整数型				
参数名	类型	参考	可空	数组	备注
被加数	整数型				
加数	整数型				
返回 (被加数 + 加数)					

调用该子程序的代码：

```
计算结果=子程序_求和(33243，98210)
```

程序代码表示调用“子程序\_求和”，该子程序有返回值，有两个参数。

第一个是“被加数”，第二个是“加数”，调用该子程序时需要提供这两个参数。如程序中需要获得计算结果，所以需要定义变量“计算结果”。但需要注意数据类型要定义为整数型，要与返回值定义的类型保持一致。

2.3.6 子程序的参数

子程序每个参数都存在“参考”、“可空”和“数组”三个属性。这三个属性决定了子程序参数的几种特性。

1) 子程序参数的“参考”属性

子程序参数的“参考”属性表示子程序的该参数在传递数据时是否为传递指向数据的地址。

在“参考”属性上按下空格键，则“参考”属性打上“√”，表示该属性为“参考”属性，如果要取消“参考”属性，再次按下空格键，会去掉“参考”属性，如图2-49所示。

子程序名	返回值类型	公开	备注		
设置服务器	逻辑型				
参数名	类型	参考	可空	数组	备注
配置名	文本型	√			

图2-49 选择“参考”属性

如果所传递过来的参数数据为数组、用户定义数据类型、库定义数据类型、文本型、字节集型数据，则无论此属性是否为“真”（选中），都将传递原数据的内存地址。

如果所传递过来数据的类型与相应位置处参数的数据类型不一致但可以相互转换。如：将“整数型”数据传递到“小数型”的参数。在数据被实际传递前，系统将首先自动将“整数型”数据转换为“小数型”数据，然后再进行传递。因此在这种情况下，即使参





考属性为“真”，系统也无法传递指向原数据的地址，只能传递数据本身。

如果系统将数据地址成功地传递过来，那么在子程序中对此参数内容的更改将会相应地反映到调用子程序时所提供的参数数据上。

例如：新建一个易程序，添加一个编辑框和一个按钮组件。

双击按钮组件，在代码编辑界面新建一个子程序，定义子程序名为“参数赋值”，为该子程序定义二个整数型参数：修改名称为“参考参数”和“非参考参数”。将“参考参数”设置为“参考”属性。代码所下：

子程序名	返回值类型	公开	备注		
参数赋值					
参数名	类型	参考	可空	数组	备注
参考参数	整数型	✓			
非参考参数	整数型				

参考参数 = 100

非参考参数 = 100

在“\_按钮1\_被单击”子程序中输入以下代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	整数型			
变量2	整数型			

参数赋值 (变量1, 变量2)

编辑框1.加入文本 (“参考参数中的变量：” + 到文本 (变量1) + #换行符, “非参考参数属性中的变量：” + 到文本 (变量2) + #换行符)

按F5键运行程序，如图2-50所示。

在程序中，每次点击按钮，都会调用“参数赋值”子程序对两个参数进行修改。从编辑框的显示结果可以看出，对应“参考参数”的变量1，因子程序内部对相应参数的操作而被修改，而对应“非参考参数”的变量2没有发生任何变化。

对上面的程序代码改为两个文本变量，子程序修改后如下：

子程序名	返回值类型	公开	备注		
参数赋值					
参数名	类型	参考	可空	数组	备注
参考参数	文本型	✓			
非参考参数	文本型				

参考参数 = “我是参考参数”

非参考参数 = “我是非参考参数”

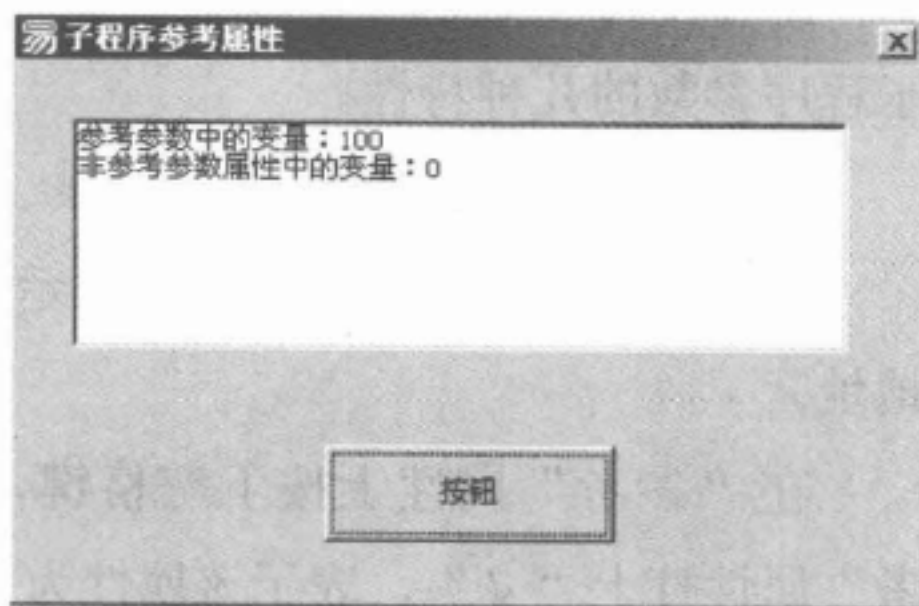


图2-50 “参考”属性对变量的影响比较





在“\_按钮1\_被单击”子程序中输入以下代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	文本型			
变量2	文本型			

参数赋值 (变量1, 变量2)

编辑框1. 加入文本 (“参考参数中的变量:” + 变量1 + #换行符, “非参考参数属性中的变量:” + 变量2 + #换行符)

按F5键运行程序，如图2-51所示。

上述程序中，每次点击按钮，就会调用“参数赋值”子程序对2个参数进行修改。从编辑框的显示结果可以看出，对应“参考参数”的变量1，因子程序内部对相应参数的操作而被修改，对应“非参考参数”的变量2因为是文本型参数，传递的是参数的地址，所以虽然没有选择参考属性，但同样被修改了。

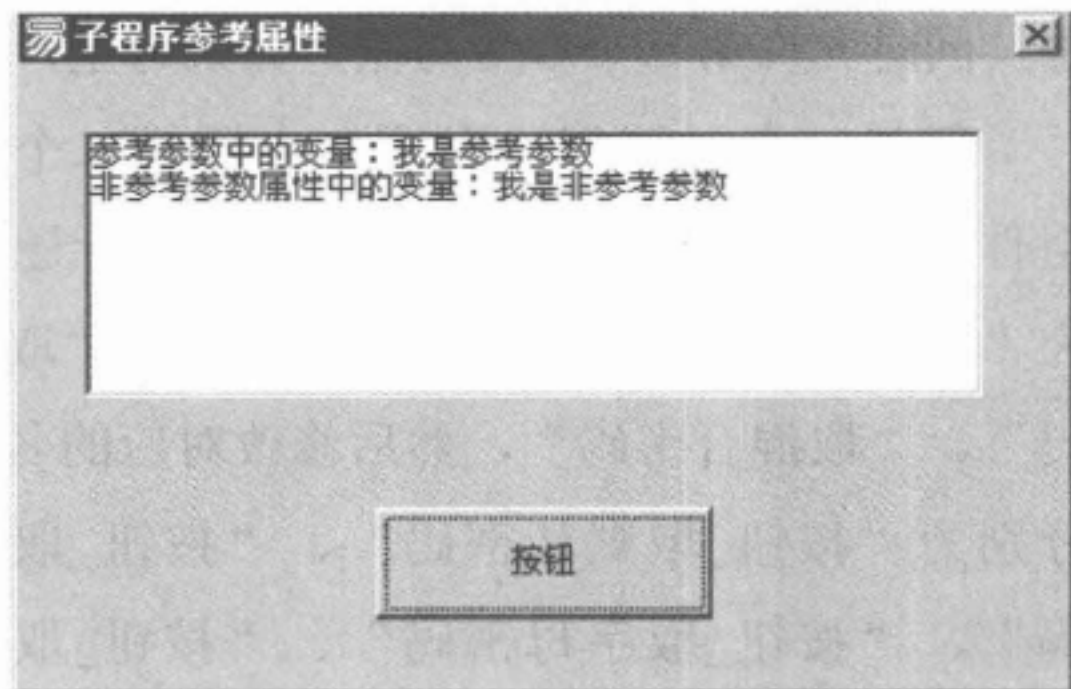


图2-51 “参考”属性对变量的影响比较

## 2) 子程序参数的“可空”属性

子程序参数的“可空”属性表示子程序的该参数在调用时是否可以不提供该参数，这样会给子程序调用时带来方便性和灵活性。

如果具有“可空”属性，表示在调用该子程序的时候可以不提供该参数，如果不具有“可空”属性，表示在调用该子程序时必须提供该参数，否则系统预编译会报错。

在“可空”属性上按下空格键，则“可空”属性打上“√”，表示该属性为“可空”属性，如果要取消“可空”属性，再次按下空格键，会去掉“可空”属性，如图2-52所示。

子程序名	返回值类型	公开	备 注		
设置服务器	逻辑型				
参数名	类 型	参考	可空	数组	备 注
配置名	文本型		√		

图2-52 选择“可空”属性

易语言自带的支持库命令中有一些参数可以被省略，查找帮助时可以看到有的参数用中括号括起来了。如“读入文本( )”命令。

命令原型为：

<文本型> 读入文本 (整数型 欲读入文本数据的文件号, [整数型 欲读入文本数据的长度] )

参数<1>的名称为“欲读入文本数据的文件号”，类型为“整数型(int)”。该文件号由“打开文件”命令所返回。



参数<2>的名称为“欲读入文本数据的长度”，类型为“整数型（int）”，可以被省略。如果本参数被省略，默认读入文件中的所有文本数据。

可以看到在“读入文本（）”命令的第二个参数“[整数型 欲读入文本数据的长度]”，即表示该参数是可以省略的，是“可空”的。如果为空，则默认读入文件中的所有文本数据。

“用户自定义子程序”也可以设置具有“可空”属性的参数。如需要检测当前子程序被调用时，该参数是否传递了数据，可用“是否为空（）”命令进行确认。

例如：制作一个取多种随机密码子程序。通过第二个参数来确定要取的是哪一种密码，同时允许第二个参数为空，如果为空，则返回一段默认密码。

新建一个易程序，在窗口中添加二个编辑框组件，四个按钮组件，将按钮组件的标题分别改为“取默认密码”、“取数字密码”、“取字母密码”、“取混合密码”，然后修改对应的名称属性分别为“按钮\_取默认密码”、“按钮\_取数字密码”、“按钮\_取字母密码”、“按钮\_取混合密码”，如图2-53所示。

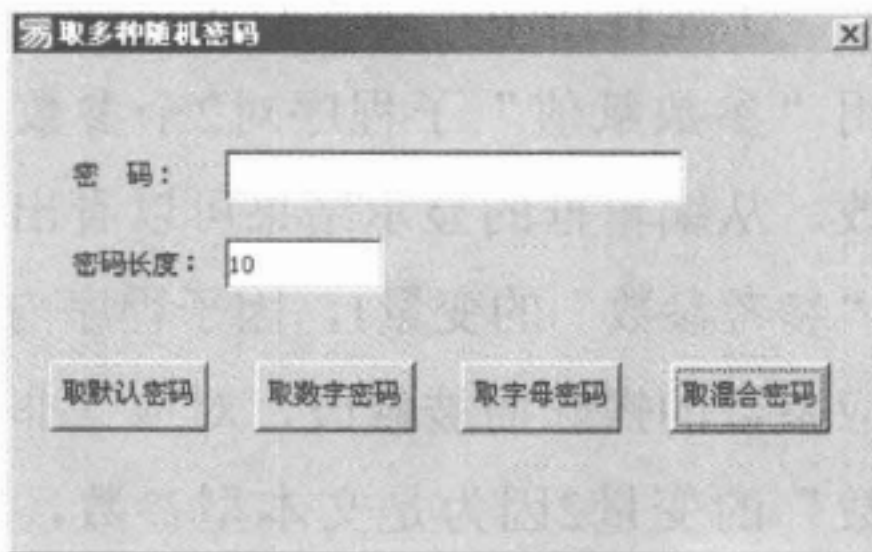


图2-53 取多种随机密码

双击启动窗口，切换到代码编辑界面，新建一个子程序，将子程序名改为“子程序\_取得随机密码”。给该子程序定义二个整数型参数，参数名分别为“参数\_位数”、“参数\_模式”，然后设置“参数\_模式”参数的“可空”属性，在子程序中输入如下程序代码：

子程序名	返回值类型	公开	备 注		
子程序_取得随机密码	文本型				
参数名	类 型	参考	可空	数组	备 注
参数_位数	整数型				
参数_模式	整数型		✓		

变量名	类型	静态	数组	备注
局部_变量	整数型			
局部_文本	文本型			

```
置随机数种子 0
判断 (参数_模式 = 1)
    计次循环首 (参数_位数, 局部_变量)
        局部_文本 = 局部_文本 + 字符 (取随机数 (48, 57))
    计次循环尾 0
判断 (参数_模式 = 2)
    计次循环首 (参数_位数, 局部_变量)
        局部_文本 = 局部_文本 + 多项选择 (取随机数 (1, 2), 字符 (取随机数 (65, 65 + 25)), 字符 (取随机数 (97, 97 + 25)))
    计次循环尾 0
判断 (参数_模式 = 3)
    计次循环首 (参数_位数, 局部_变量)
        局部_文本 = 局部_文本 + 多项选择 (取随机数 (1, 3), 字符 (取随机数 (65, 65 + 25)), 字符 (取随机数 (97, 97 + 25)), 字符 (取随机数 (48, 57)))
    计次循环尾 0
判断 (是否为空 (参数_模式) = 真)
    局部_文本 = "我爱易语言"
返回 (局部_文本)
```



子程序中，“参数\_模式”参数控制子程序返回何种密码，1为数字密码，2为字母密码，3为数字和字母的混合密码，该参数为空则返回默认密码：“我爱易语言”。

回到“\_启动窗口”双击按钮“取默认密码”，在“\_按钮\_取默认密码\_被单击”子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_按钮_取默认密码_被单击			

编辑框\_密码.内容 = 子程序\_取得随机密码 (到整数 (编辑框\_密码长度.内容), )

可以看到，在“\_按钮\_取默认密码\_被单击”子程序中，省略了“子程序\_取得随机密码”子程序第2个参数的填写。

双击按钮“取数字密码”，在“\_按钮\_取数字密码\_被单击”子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_按钮_取数字密码_被单击			

编辑框\_密码.内容 = 子程序\_取得随机密码 (到整数 (编辑框\_密码长度.内容), 1)

可以看到，在“\_按钮\_取数字密码\_被单击”子程序中，“子程序\_取得随机密码”子程序第2个参数设置为1。依此编写“\_按钮\_取字母密码\_被单击”，“\_按钮\_取混合密码\_被单击”子程序，分别调用“子程序\_取得随机密码”子程序，第2个参数设置为2和3。

按F5键运行程序。点击按钮“取默认密码”，返回的是默认密码。因为程序运行的“子程序\_取得随机密码”子程序第2个参数没有填参数，如图2-54（a）所示。

点击按钮“取混合密码”，在程序运行的“子程序\_取得随机密码”子程序第二个参数设置参数为3，因此返回的是数字和字母的混合密码。结果如图2-54（b）所示。

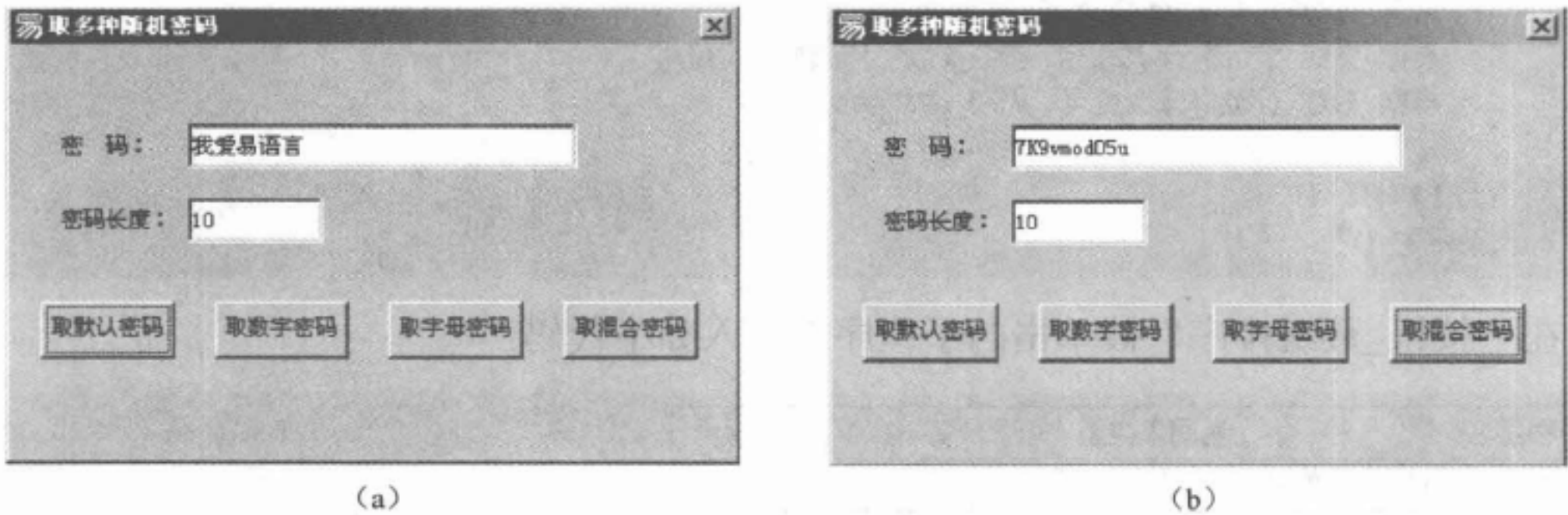


图2-54 取多种随机密码

3) 子程序参数的“数组”属性

子程序参数的“数组”属性表示子程序的参数是一个数组变量。

在“数组”属性上按下空格键，则“数组”属性打上“√”，表示该属性为“数组”属性，如果要取消“数组”属性，再次按空格键即可，如图2-55所示。



子程序名	返回值类型	公开	备注		
设置服务器					
参数名	类型	参考	可空	数组	备注
配置名	文本型			✓	

图2-55 选择“数组”属性

从程序参数“参考”属性的学习我们知道，参数如果是“数组”型变量，就自动具有“参考”属性。所以在子程序中对此参数内容的更改将会相应地反映到调用子程序时所提供的参数数据上。根据这个特点编写一个数组排序的小例子，来了解数组参数的作用。

【例】新建一个易程序，在窗口中添加一个编辑框和一个按钮组件，修改编辑框内容为：“56, 4, 99, 38, 27, 19, 47, 78, 91, 41”，修改按钮名称为：“按钮\_数组排序”。

在代码编辑界面新增一个子程序，修改子程序名为“子程序\_数组排序”，定义一个该子程序参数，将参数的名称改为“参数\_数组”，并将参数的数组属性选中，输入如下代码：

子程序名	返回值类型	公开	备注
子程序_数组排序			

参数名	类型	参考	可空	数组	备注
参数_数组	整数型			✓	

变量名	类型	静态	数组	备注
局部_计次1	整数型			
局部_计次2	整数型			
局部_临时存放	整数型			

```

--> 计次循环首 (取数组成员数 (参数_数组), 局部_计次1)
  --> 计次循环首 (取数组成员数 (参数_数组), 局部_计次2)
    --- 如果真 (参数_数组 [局部_计次1] < 参数_数组 [局部_计次2])
      局部_临时存放 = 参数_数组 [局部_计次2]
      参数_数组 [局部_计次2] = 参数_数组 [局部_计次1]
      参数_数组 [局部_计次1] = 局部_临时存放
    --- 计次循环尾 0
  --- 计次循环尾 0
  
```

在“按钮\_数组排序”被单击的子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_按钮_数组排序_被单击			

变量名	类型	静态	数组	备注
数组	数组类型		0	
局部_计次	整数型			

```

数组 = { 56, 4, 99, 38, 27, 19, 47, 78, 91, 41 }
子程序_数组排序 (数组)
编辑框_数组.内容 = ""
--> 计次循环首 (取数组成员数 (数组), 局部_计次)
  编辑框_数组.内容 = 编辑框_数组.内容 + 到文本 (数组 [局部_计次]) + ","
--- 计次循环尾 0
  
```



按F5键运行程序。程序运行结果如图2-56所示。

点击“数组排序”按钮进行数组排序将重新显示该组数字。结果如图2-57所示。

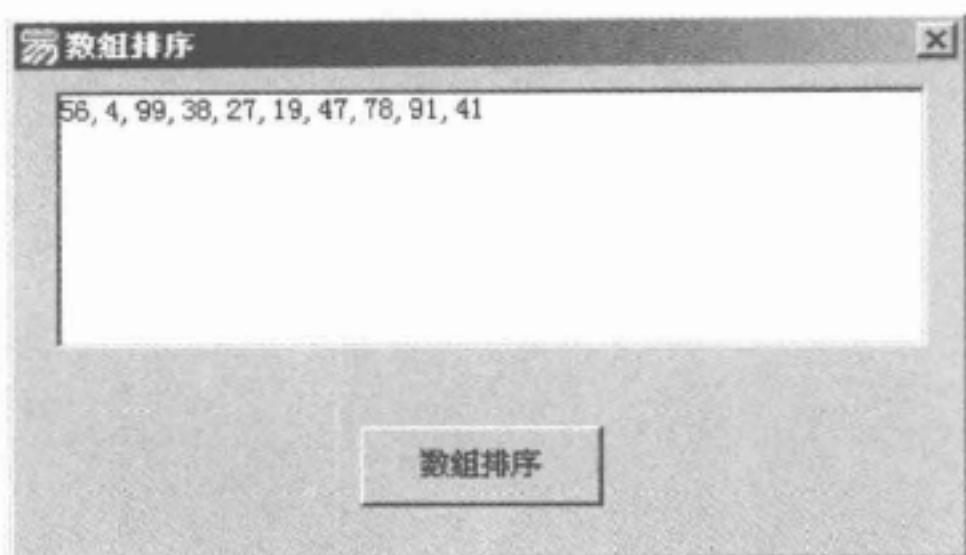


图2-56 数组排序例程排序前

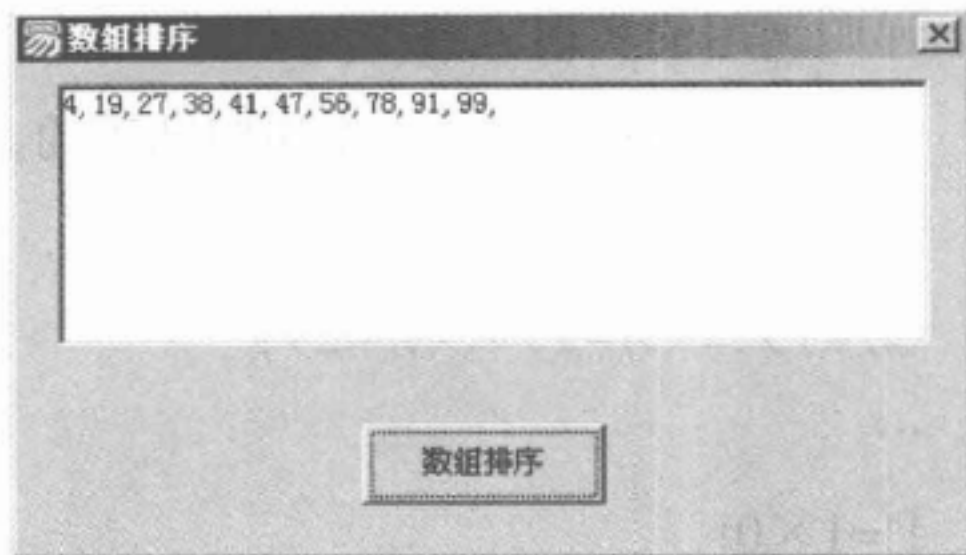


图2-57 数组排序例程排序后

可以看到编辑框显示出调用子程序前后数组变量中各成员的值，在子程序中对数组参数的更改会相应地反映到调用子程序时所提供的数组参数上。

### 2.3.7 子程序的递归调用

递归调用是子程序的一种特殊的调用方法。

子程序一般情况下都是一个子程序调用另外一个子程序，此前介绍的子程序都是这种情况。子程序也可以自己调用自己，这种情况称为“递归调用”。

递归调用有两种方式：一种是在调用一个子程序的过程中，又调用了该子程序自身，这种方式称为直接递归；另一种是在调用一个子程序（假定为子程序1）的过程中，该子程序调用另外一个子程序（假定为子程序2），而在子程序2中又调用子程序1，这种方式称为间接递归。

在实际编程中，一些问题采用递归调用的方法来解决更为直观、简便，而且大多数采用的都是直接递归的方式。

求正整数的阶乘问题，如 $5!$ 可化为 $5 \times 4!$ ，而 $4!$ 又可化为 $4 \times 3!$ ，…， $1!$ 可以化为 $1 \times 0!$ ， $0! = 1$ ，于是 $5!$ 便可以表示为 $5 \times 4 \times 3 \times 2 \times 1 \times 1$ ，其值为120。

通过上述求阶乘的例子，可以看到使用递归调用解决问题的特点是，将原有的比较复杂的问题分解为一个新的比较简单的问题，而新问题又要用原有问题的解决方法再进行分解，这便出现了递归。按照这一特点将问题继续分解下去，每次出现的新问题都是原问题简化的子问题，而最终分解出来的新问题是具有已知解的最简单问题。这就是有限的递归调用。

**注解：**编程中使用递归方法在程序执行起来内存开销比较大，既要花费较长的时间，又要占用较多的内存单元，所以一般不提倡用递归调用。但使用递归调用方法编写的程序简洁清晰、可读性强，有的问题甚至只能用递归解决。因此，递归算法是编程中的一个重要算法之一。



在使用递归调用时，必须有一个明确的递归结束条件，无条件递归调用将会成为死循环而不能正常结束。

递归调用的实现方法

以求 $n!$ 为例，说明实现递归调用的方法。

由于 $n! = n(n-1)!$

$(n-1)! = (n-1) \times (n-2)!$

...

$1! = 1 \times 0!$

因此，可以抽象得到一个求 $n!$ 的递推公式：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

有了这个递推公式，就很容易写出实现求阶乘的递归子程序。

通过一个例程来具体看一下。

**【例】** 点击按钮，在标签中显示编辑框中所输入数的阶乘结果。

新建一个易程序，添加两个标签，一个编辑框，一个按钮组件，如图2-58所示。

根据介绍的递推公式分析，该子程序会递归的调用自身。输入代码如下：

子程序名	返回值类型	公开	备注
求阶乘	长整数型		

参数名	类型	参考	可空	数组	备注
n	整数型				

--- 如果 (n = 0)  
--- 返回 (1)  
--- 返回 (n × 求阶乘 (n - 1))

在按钮1的单击事件中输入如下代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

标签2.标题 = 编辑框1.内容 + “的阶乘等于：” + 到文本 (求阶乘 (到整数 (编辑框1.内容)))

按F5键运行程序。在编辑框中输入20，点击“开始计算”按钮查看结果，如图2-59所示。

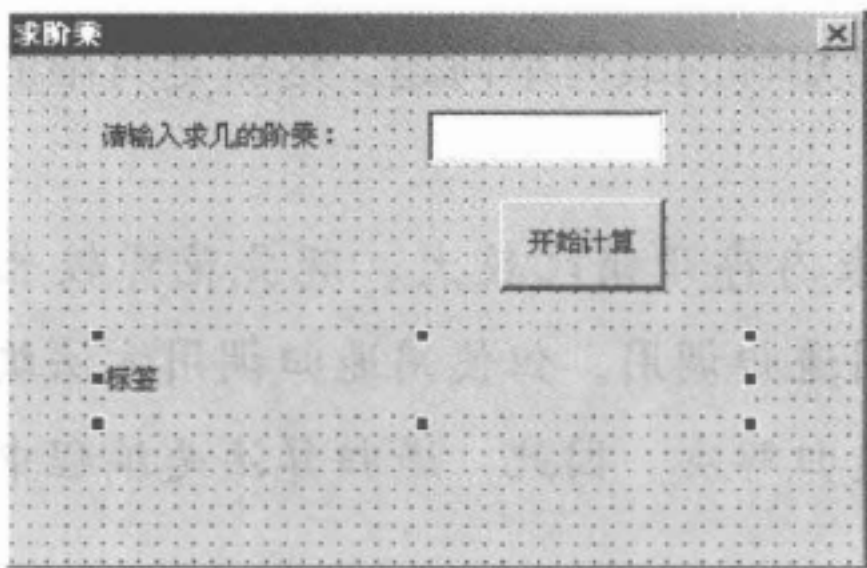


图2-58 求阶乘程序界面

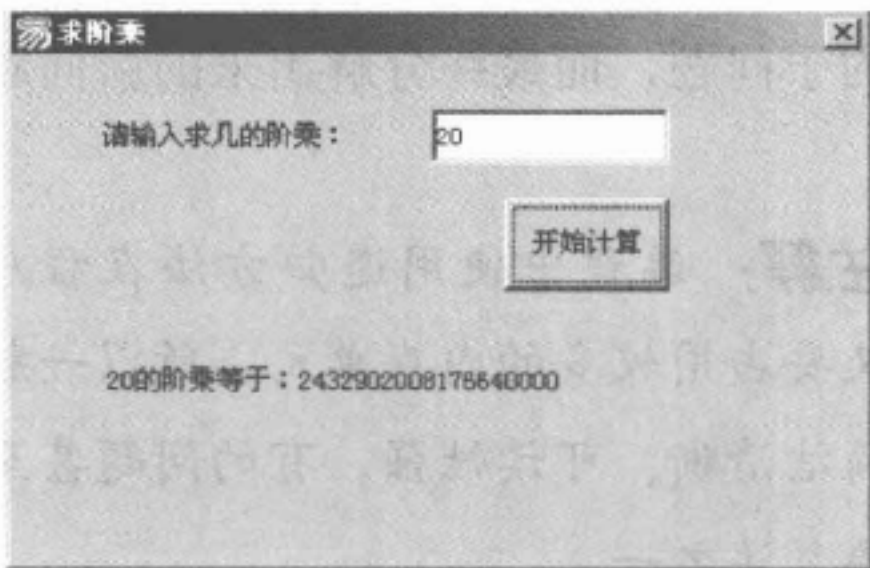


图2-59 求阶乘程序运行效果



**注解：**在输入大于20的数的时候，结果会显示出负值，这是因为结果已经超出了定义的“长整数型”的返回值范围，这里可以将返回值设置为“双精度小数型”来满足计算的要求。

再举个递归调用子程序的例子。

**【例】**在程序中，经常要列出指定目录下的所有文件（包括子目录下的文件），这就可以用递归调用来实现。

首先从指定的某个目录开始，假设为D:\document1，需要使用一个磁盘操作命令“寻找文件（）”，用来寻找出指定目录下的文件或目录名称，通过两个循环可以分别遍历出指定目录下的文件和目录名称。但是，要实现将指定目录下的所有子目录下的文件也遍历出来，显然还缺少对子目录内文件和子目录下的一级子目录的遍历。而对于子目录的遍历同样要执行一下刚才的操作，即遍历该子目录下的文件和该子目录下的一级子目录，对于寻找到的每一个子目录都有相同的操作要作。因此很明显，这一部分功能需要使用递归调用。但如何在程序中确立结束递归的出口，避免陷入无限的循环中去呢？这里可使用“寻找文件（）”命令，当寻找不到新的文件或目录时会返回一个空文本，可以根据这个特点来控制递归的出口。

“寻找文件（）”命令有自己的使用特点和注意事项，在“3.13 磁盘操作命令”一节中会详细介绍，本处主要介绍递归的思想。

下面就开始具体的代码编写。

第一步：新建一个易程序，添加两个编辑框、两个标签、两个按钮、一个通用对话框、一个列表框。并设置相应的名称和标题，如图2-60所示。

新建的寻找文件的子程序代码如下。



图2-60 寻找文件界面设计

子程序名	返回值类型	公开	备 注		
子程序_寻找文件					
参数名	类 型	参考	可空	数组	备 注
参数_目录名	文本型				
参数_文件名或类型	文本型		✓		

变量名	类型	静态	数组	备注
寻找结果	文本型			

```

-- 如果真 (参数_目录名 = "")
    信息框 ("请选择欲寻找的目录", #错误图标, "错误")
    返回 ()
-- 如果真 (参数_文件名或类型 = "")
    参数_文件名或类型 = "*. *"
-- 如果真 (取文本右边 (参数_目录名, 1) ≠ "\")
    参数_目录名 = 参数_目录名 + "\"
寻找结果 = 寻找文件 (参数_目录名 + 参数_文件名或类型, )
    
```





```

--> 判断循环首 (寻找结果 ≠ "")
    列表框_显示文件. 加入项目 (参数_目录名 + 寻找结果, )
    寻找结果 = 寻找文件 ( )
-- 判断循环尾 ()
寻找结果 = 寻找文件 (参数_目录名 + "*. *", #子目录)
--> 判断循环首 (寻找结果 ≠ "")
    -- 如果真 (寻找结果 ≠ "." 且 寻找结果 ≠ "..")
        子程序_寻找文件 (参数_目录名 + 寻找结果, 参数_文件名或类型)
        处理事件 ()
    寻找结果 = 寻找文件 ( #子目录)
-- 判断循环尾 ()

```

第二步：在这个子程序中，首先遍历传递过来的“参数\_目录名”路径下的所有文件名，在文件名遍历完毕之后，遍历“参数\_目录名”路径下的所有子目录名，在寻找到子目录之后，就采用递归调用该子程序本身。语句为：

子程序\_寻找文件 (参数\_目录名 + 寻找结果, 参数\_文件名或类型)

程序的运行效果如图2-61所示。



图2-61 寻找文件程序运行效果

## 2.4 我的播放器（二）

在第二章中，我们学习了易语言编程基础知识。在本节，根据本章讲到的易语言的资源表，现在为MP3播放器程序添加底图和背景音乐。

新建一个易语言程序，设计程序界面。添加一个按钮。将“启动窗口”的标题改为“我的播放器（二）”。将“启动窗口”的“底图方式”属性设置为“缩放图片”。以便我们设置的图片会完整的显示。将“按钮1”的标题设置为：“播放按钮”，“按钮1”的名称设置为：“按钮\_播放”，界面效果如图2-62所示。

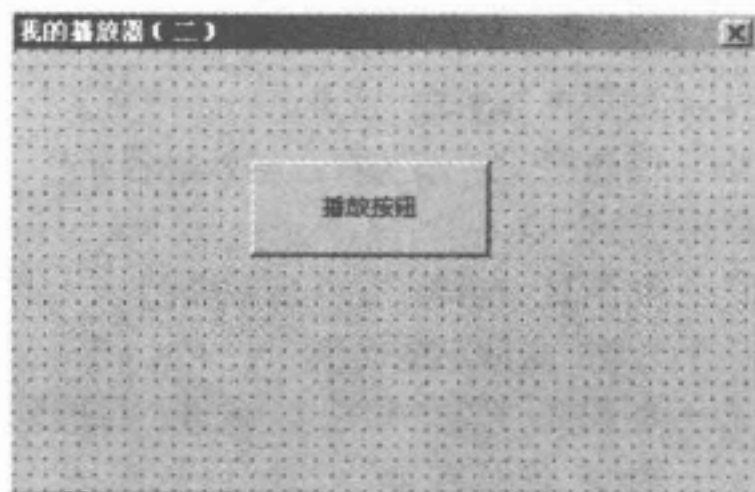


图2-62 设置按钮





下面来编写代码。双击“播放按钮”进入代码编辑区。在“播放按钮”的单击事件中输入：

子程序名	返回值类型	公开	备注
按钮_播放_被单击			
_启动窗口.背景音乐 = { } 播放MP3 (1, “在路上.mp3” )			

表示停止启动窗口的背景音乐，播放当前目录下的“在路上.mp3”文件。  
使用本章学到的方法，在资源表中添加一张图片和一首背景音乐，如图2-63所示。

图片或图片组名称	内容	公开	备注
图片	308540		

声音名称	内容	公开	备注
音乐	8589		

图2-63 添加图片和音乐

**注解：**声音资源只支持“\*.mid”和“\*.wav”格式的声音文件。

双击启动窗口界面，生成“\_启动窗口\_创建完毕”子程序，在该子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			
_启动窗口.底图 = #图片 _启动窗口.背景音乐 = #音乐			

这样，程序启动后，背景音乐和底图就有了，运行的效果如图2-64所示。



图2-64 MP3效果

## 2.5 小结

本章介绍了易语言中数据类型的分类，变量、常量、资源的定义和使用方法，并介绍了易语言中运算符和表达式的应用，最后介绍了子程序的编写与调用，并编写了递归调用的例子。





## 2.6 习题

2-1 易语言的数据类型从数据结构来区分,可分为\_\_\_\_\_和\_\_\_\_\_。

2-2 易语言中基本数据类型有6种,包括数值型、逻辑型、\_\_\_\_\_,文本型、\_\_\_\_\_,子程序指针型。

2-3 数值型数据包括:字节型、\_\_\_\_\_,整数型、长整数型、\_\_\_\_\_,双精度小数型。

2-4 易语言的短整数型数据类型的取值范围为\_\_\_\_\_到\_\_\_\_\_之间,存储空间为\_\_\_\_\_字节。

2-5 易语言中的运算符根据操作的性质进行分类,可以分为\_\_\_\_\_,关系运算符、逻辑运算符、\_\_\_\_\_。

2-6 易语言的赋值运算符是\_\_\_\_\_。

2-7 运算符的优先级可以通过\_\_\_\_\_来改变。

2-8 变量有自己的作用域范围。从变量的使用范围来区分,可以将变量分为\_\_\_\_\_,和\_\_\_\_\_。从变量的属性来区分,还可以分为\_\_\_\_\_和\_\_\_\_\_。

2-9 易语言常量的类型可以为\_\_\_\_\_,文本型、逻辑值、\_\_\_\_\_和长文本型。

2-10 常量只能在\_\_\_\_\_时定义,在\_\_\_\_\_过程中是不能修改其值的。

2-11 在易语言中,可以存放程序中用到的\_\_\_\_\_,\_\_\_\_\_等二进制数据资源。

2-12 易语言中的子程序可以分为两大类:“事件子程序”和“\_\_\_\_\_”。

2-13 易语言事件子程序的名称、返回值类型和参数个数都是\_\_\_\_\_定义的,不允许用户任意修改。

2-14 “用户自定义子程序”是由用户创建,其\_\_\_\_\_个数和\_\_\_\_\_都由用户自行定义,用户可以根据需要在程序设计时对其任意修改。

2-15 子程序每个参数都存在“参考”、“\_\_\_\_\_”和“\_\_\_\_\_”三个属性。

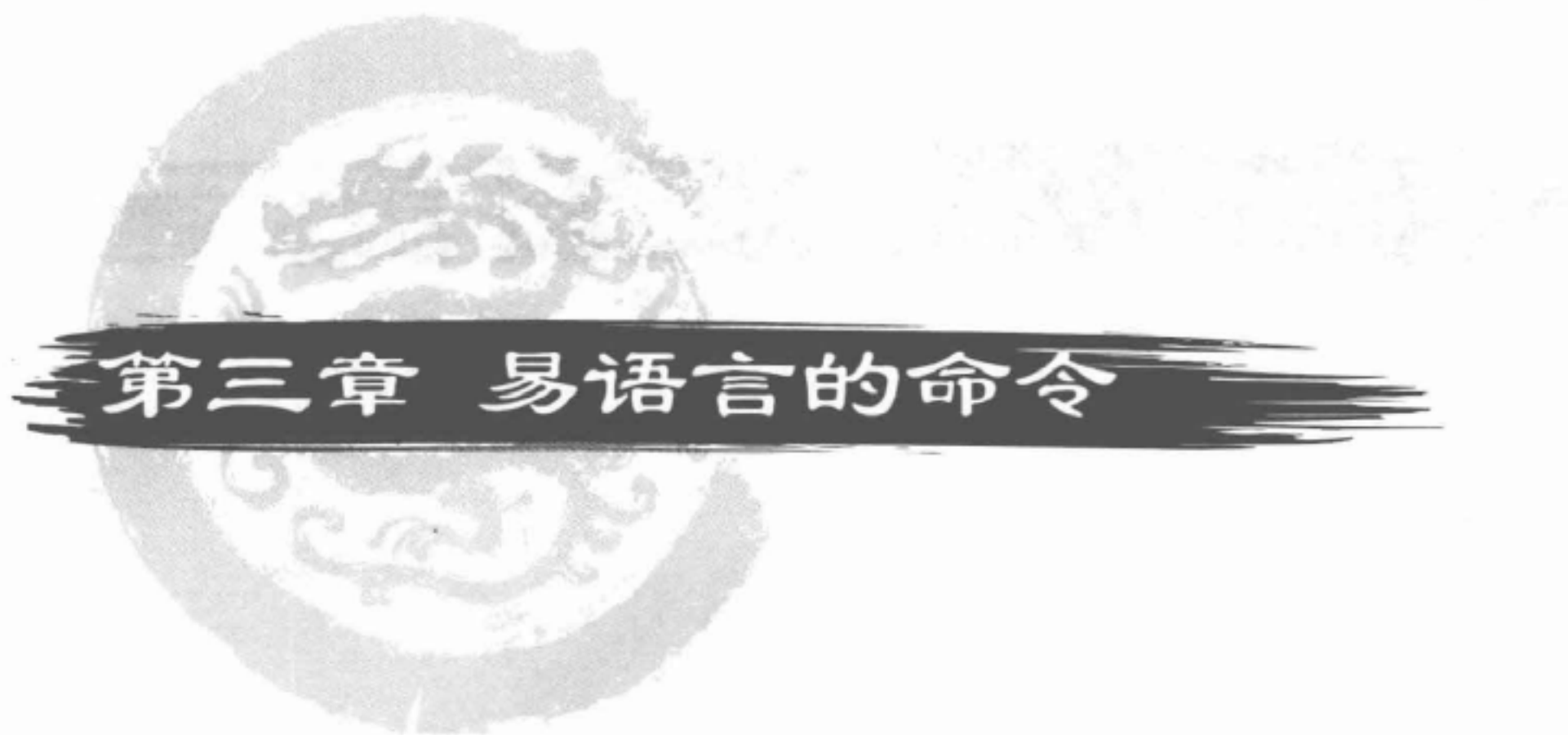




## 第二部分 易语言的命令与组件

命令与组件的使用是现代可视化编程工具学习中重要的目标之一。在本部分我们将详细学习如何使用易语言中所提供的核心命令与基本窗口组件。在学习完本部分后，大家基本可以自行完成常用的小型应用程序的设计与编写。





## 第三章 易语言的命令

### 本章目标

在本章结束时，我们能够：

- 了解命令
- 掌握易语言命令的特性
- 掌握易语言命令的使用
- 了解易语言核心库所提供的命令



为了达到使用计算机进行计算和解决问题的目的，必须将问题解决方案用计算机指令表达出来。换句话说，必须提供给计算机一套指令，使其能方便地解决问题，这一套指令就是一段程序。一旦给出解决某问题的程序，那么这个程序可以任意多次地运行来解决相同的问题。将解决问题的一段程序进行封装既为命令。一个命令可以由一个或多个简单的命令来封装成一个功能更强大的命令。这样，在需要相同作用的功能时只要调用新的命令既可以得到相同的结果，这会大大简化编程的复杂程度并缩短软件的开发周期。一个软件是由各种命令组合而成的，不同的命令完成不同的工作。在易语言中提供了类似大量的命令，用户可以使用这些命令来实现预想的运行效果。命令是软件的基本组成部分，要学习用易语言来编写软件，首先就要了解易语言所提供的命令。

## 3.1 了解易语言命令

### 3.1.1 易语言命令概述

命令：一个功能调用的开始。

命令的参数：欲调用一个功能时附加的数据或条件。（有时也用来获得调用后的结果）

命令的返回值：用来获得调用一个功能后所得到的反馈。

用生活中买东西的简单过程来模拟一下命令的调用过程。

假设：我们知道所要购买物品的单价，需要告诉售货员物品的数量，并付对应的货款，售货员收取货款，并把对应数量的物品给我们。购物的过程就结束了。

在这个例子中，知道所要购买物品的单位售价，相当于了解一个命令的使用方式；告诉售货员所需物品的数量、并付钱，钱和物品数量就相当于调用命令所传递的参数；售货员把物品给顾客，就相当于命令给我们的返回值。

### 3.1.2 易语言命令的格式

易语言中调用一个命令的完整格式如下：

[返回值][所属对象.]命令名称 ([参数1], [参数2], ...)

**注解：** []中代表不是必须存在的部分。

易语言中的大多数命令格式如下：

命令名称 (参数, ...)

### 3.1.3 易语言命令的参数

命令的参数可以是一个数据类型的具体数据值，也可以是一个变量。但要注意的是，





在调用时传递给命令的参数必须满足该命令对参数的定义，否则会出现语法错误。

大部分命令都需要填写参数，参数是用括号括起来的，多个参数间用逗号分隔。部分命令无需参数，但括号不能省略，如“结束（）”命令。各种命令所要求参数的个数以及数据类型各有不同，由其定义所限定。例如“到文本（）”命令，该命令只需一个参数，参数内容为欲转换成文本的数据。有些命令的参数很多，如“子文本替换（）”命令的格式如下：

子文本替换（欲被替换的文本，欲被替换的子文本，[用作替换的子文本]，[进行替换的起始位置]，[替换进行的次数]，是否区分大小写）

另外有些命令的参数拥有“参考”属性。如果在调用这样的命令时，为拥有参考属性的参数提供一个变量，则这个变量的内容有可能被所调用的命令改变。

**注解：**易语言中的命令有很多，不需要将全部的命令语法和参数含义都背下来。在实际开发工作中，可以使用命令分步输入方法（将语句展开），根据参数提示输入，或按F1键查看易语言的即时帮助，或查阅帮助文档。

### 3.1.4 易语言命令的返回值

大多数命令执行完毕后都有返回值。有的命令返回运算结果，如“求正弦（）”命令，返回求得的正弦值；有的命令返回执行后的反馈信息，如“取文本左边（）”命令，返回取出来的文本内容；有的命令返回运行是否成功的状态，如“创建目录（）”命令，创建成功则返回“真”，创建失败则返回“假”等。大部分时候，当前命令的返回值对后续命令来说非常重要。当一个命令运行成功了，就弹出信息框提示成功，否则提示失败，如代码所示：

```
-- 如果 (创建目录 ("C:\易语言") = 真)
-- 信息框 ("创建成功", 0, )
-- 信息框 ("创建失败", 0, )
```

各命令的语法规则了其返回值的数据类型，在实际使用中，应当注意有可能需要对返回值的数据类型加以转换，如：编辑框的内容属性只接收文本型，因此要显示一个数字就需要使用“到文本（）”命令将数字转换为文本形式显示，代码如下所示：

编辑框1.内容 = 到文本 (取绝对值 (-50))

“取绝对值（）”命令的返回值是一个数值型的，如果要以文本方式显示在编辑框中，就要用“到文本（）”命令进行转换。

**注解：**“到文本（）”和“取绝对值（）”命令将在本章后面讲解，编辑框的使用将在第四章中讲解。

有些命令的返回值是一个通用型的数据，代表根据情况不同，其返回值的数据类型也不同。例如“多项选择（）”命令。该命令有2个参数（参数还可增加），第一个参数是





索引值，第二个参数及后续参数是待选项，待选项可以重复添加。待选项数据类型是通用型（表示第二个及后续参数可以是任意数据类型）的，返回哪个待选项取决于第一个参数的索引值。索引值是1，则返回第一个待选项；索引值是2，则返回第二个待选项……。所以，返回值类型为具体返回数据所对应的类型。

【例】 通过编写一个例程来了解一下“多项选择（）”命令。

第一步：新建一个易程序，在窗口中添加一个“画板”组件和一个“按钮”组件。

第二步：修改“按钮”名称为：按钮\_多项选择；修改“画板”名称为：画板\_多项选择，双击按钮组件，在“\_按钮\_多项选择\_被单击”事件子程序中输入如下代码：

子程序名	返回值类型	公开	备注
_按钮_多项选择_被单击			

+ 画板\_多项选择.滚动写行 (多项选择 (3, “易语言”, 3.14, [2009年1月19日]))

第三步：运行程序，点击按钮，可以看到画板中显示出了第3个待选项日期时间型的[2009年1月19日]，如图3-1所示。

注解：关于“多项选择（）”命令将在本章的后面详细介绍。

有些命令无返回值，如“销毁（）”命令，此类无返回值的命令运行后不返回任何值，所以直接使用即可。

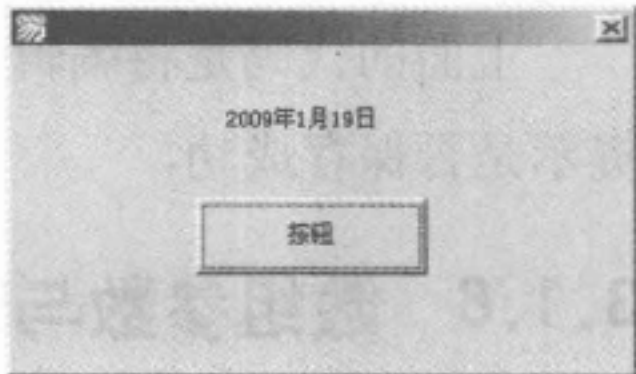


图3-1 运行结果

子程序名	返回值类型	公开	备注
_按钮_销毁_被单击			

销毁 0

注解：命令是否有返回值，返回值是什么数据类型，都可以通过易语言的即时帮助系统查找到，在程序编辑界面，将光标停在欲查询的命令上，然后按下F1键，可以在提示面板中看到该命令的帮助。在提示面板中“调用格式”一行，写在命令名前面的就是该命令的返回值类型，如果无返回值则显示“无返回值”，如图3-2所示。

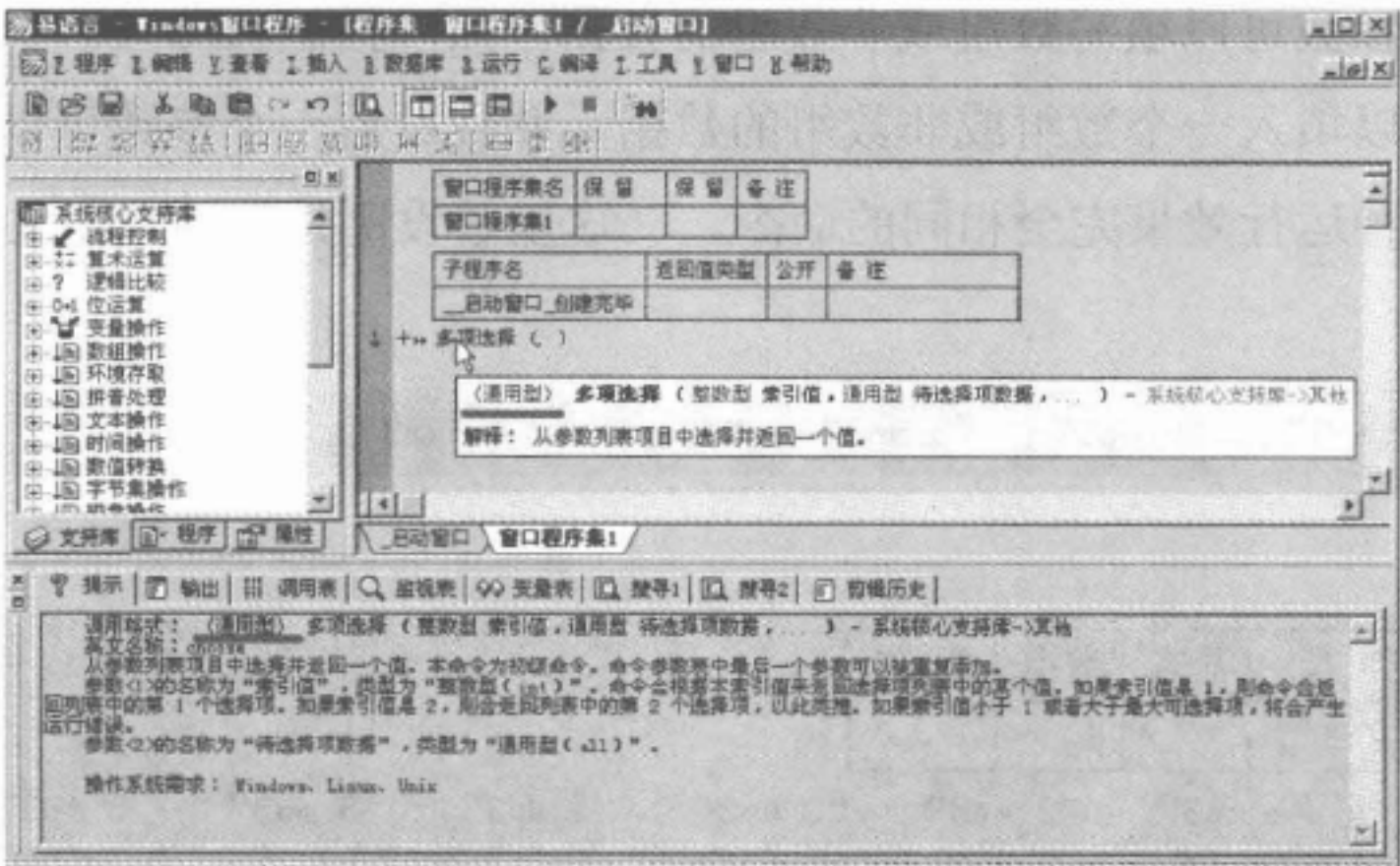


图3-2 查看命令的返回值类型





### 3.1.5 易语言命令嵌套调用

易语言中的命令是可以嵌套使用的，即命令中的参数是另一个命令的返回值。如：

编辑框1.内容=到文本（到数值（编辑框1.内容）+1）

此行代码在“到文本（）”命令中套用了“到数值（）”命令，上面的代码首先会将编辑框中的内容转换成数值型，然后将转换后的数值加1，将加1的结果转换成文本重新显示在编辑框中。

如：“写到文件（）”命令的参数中套用多个命令代码如下：

```
如果（写到文件（取运行目录 0 + “\易语言.txt”，到字节集（编辑框1.内容））= 真）
  信息框（“保存成功”，0，）
信息框（“保存失败”，0，）
```

上面的代码是将编辑框中的内容保存为程序运行目录下的一个文本文件，弹出信息框提示是否保存成功。

### 3.1.6 数组参数与数组返回值

#### 1) 数组型参数

有些命令的参数必须是一个数组型变量，如“重定义数组（）”命令，该命令可以重新定义指定数组的维数及各维的上限值。该命令的第一个参数必须指定一个数组型变量。系统支持库中数组操作类命令的第一个参数，都要求填写欲操作的数组名，如：

变量名	类型	静态	数组	备注
数组变量	整数型		0	

↓ + 重定义数组（数组变量，假，3）  
信息框（取数组成员数（数组变量），0，）  
加入成员（数组变量，10）

有些命令的参数可以填写数组或非数组的数据，如“播放MP3（）”命令，该命令的第二个参数就可以填入一个数组或非数组的数据，并且最后一个参数可以重复添加，这样就产生了下面两种运行效果完全相同的命令。（这里假设所使用的MP3文件存在）

第一种：

+ 播放MP3（1，“1.mp3”，“2.mp3”，“3.mp3”，“4.mp3”，“5.mp3”，“6.mp3”）

第二种：

变量名	类型	静态	数组	备注
播放列表变量	文本型		6	

+ 播放列表变量 = { “1.mp3”，“2.mp3”，“3.mp3”，“4.mp3”，“5.mp3”，“6.mp3” }  
+ 播放MP3（1，播放列表变量）





这两种命令参数赋值的方式各有优点。对于数组型参数，可以动态的管理数组，但会占用系统内存。

2) 数组型返回值

有些命令的返回值是一个数组型变量，如：“分割文本（）”命令。此类命令需要一个数组型变量存放命令的返回值。例如：

变量名	类 型	静态	数组	备 注
分割后文本	文本型		0	

分割后文本 = 分割文本 (编辑框1.内容, “,”, )

“分割后文本”数组变量存放分割文本命令所返回的文本型数组内容。此类命令会自动重定义数组维数和成员数量，所以将变量在定义时数组成员设置为 0 即可。分割后的文本可以用一个循环将所有子文本显示出来，代码如下所示：

变量名	类 型	静态	数组	备 注
分割后文本	文本型		0	
已循环次数	整数型			

--- 计次循环首 (取数组成员数 (分割后文本), 已循环次数)  
编辑框1.加入文本 (分割后文本 [已循环次数], #换行符)  
--- 计次循环尾 0

**注解：**代码中用到的“计次循环首（）”命令会在下面讲流程控制类命令时介绍。

3.2 流程控制命令

3.2.1 了解流程控制类命令

流程控制类命令在易语言中是非常重要的一类命令，可以控制程序运行的路线，如在满足一定条件时运行一些代码，在不满足条件时运行另一些代码等。大多数程序的编写都离不开这类命令。

流程控制命令分为三类：分支类流程控制命令、循环类流程控制命令和跳转类流程控制命令。

分支类流程控制命令包括：如果（）、如果真（）、判断（）。

循环类流程控制命令包括：判断循环首（）、循环判断首（）、计次循环首（）、变量循环首（）。

跳转类流程控制命令包括：到循环尾（）、跳出循环（）、返回（）、结束（）。





**注解：**有时写了一段流程命令程序代码，发现可以更加优化流程命令，或用其他流程命令代替，如果这时重写，将会浪费很多时间。易语言提供“流程转换”功能，可以在各流程命令之间转换。

例如：写“如果（）”命令时，所得条件只有一个，可以用“如果真（）”代替，这时就可以转换。

在流程控制类命令中，除跳转类流程控制命令，都可以在流程控制语句上使用右键弹出菜单来互相转换，也可以在菜单中选择“转换为”→欲转换的流程控制语句。有些代码转换后运行结果相同。

例如：“如果（）”命令和单个“判断（）”命令可互相转换。

因为易语言提供程序流程线，所以能很清楚地观察流程控制类命令的运行路线。

例如：“如果（）”命令，易语言会清楚的指示满足条件和不满足条件时程序的执行流程。如下图，若满足条件则继续向下执行，若不满足条件则按流程线箭头所示跳到不满足条件时的执行处继续执行。

```

-- 如果 (变量1 > 变量2)
-- 编辑框1.内容 = 到文本 (变量1)
-- 编辑框1.内容 = 到文本 (变量2)

```

### 3.2.2 分支类流程控制命令

#### 1) “如果（）”命令

命令原型为：

<无返回值> 如果（逻辑型 条件）

“如果（）”命令的参数“条件”为一个逻辑型数据（非真即假）。若条件为真，则程序顺序执行后续代码；若条件为假，则程序跳转到沿虚线箭头所示的代码行继续运行。

**【例】**新建一个易程序，在“\_启动窗口”中添加一个编辑框组件和一个按钮组件，然后双击按钮，在“\_按钮1\_被单击”子程序中输入如下代码：

```

-- 如果 (编辑框1.内容 = "")
-- 编辑框1.内容 = "你好易语言"
-- 编辑框1.内容 = "我爱易语言"

```

按F5键运行程序，此时编辑框中没有任何内容，单击按钮，可以看到编辑框显示“你好易语言”；再单击按钮，此时编辑框中已有内容，则编辑框显示“我爱易语言”。

本例程源码见随书光盘中“\图书例程\第三章\流程控制\如果命令演示.e”

#### 2) “如果真（）”命令

命令原型为：

<无返回值> 如果真（逻辑型 条件）





“如果真（）”命令从流程线上就可以看出与“如果（）”命令的不同，“如果真（）”命令在条件成立的时候运行“如果真（）”命令下的代码，否则“如果真（）”命令没有任何动作。例如：

```

-- 如果真 (1 > 3)
-- 信息框 (“这是不可能滴~~”, 0, )

```

当1大于3，显示一个信息框，否则直接跳转到流程线箭头所指的位置继续运行。

### 3) “判断（）”命令

命令原型为：

<无返回值> 判断 (逻辑型 条件)

本命令根据提供的逻辑参数的值，来决定是否改变程序的执行流程，如果提供的逻辑参数值为“真”，则程序继续顺序向下执行，否则跳转到下一分支继续判断。

“判断（）”命令主要用于条件的分支选择，如下面2段代码运行效果相同，程序结构也相同，但使用“判断（）”命令，代码流程结构要清晰许多，而使用“如果（）”命令，不仅会使程序嵌套太多，程序代码难以理解，同时也降低了程序运行效率。

用“如果（）”命令编写的代码如下所示：

变量名	类型	静态	数组	备注
变量1	整数型			

```

变量1 = 2
-- 如果 (变量1 < 1)
-- 信息框 (“变量1的内容比1小”, 0, )
-- 如果 (变量1 > 1)
-- 信息框 (“变量1的内容比1大”, 0, )

```

用“判断（）”命令编写的代码如下所示：

变量名	类型	静态	数组	备注
变量1	整数型			

```

变量1 = 2
-- 判断 (变量1 < 1)
-- 信息框 (“变量1的内容比1小”, 0, )
-- 判断 (变量1 > 1)
-- 信息框 (“变量1的内容比1大”, 0, )

```

所有分支类流程控制命令的判断条件都可以使用“且”和“或”来连接多个条件，并最终返回一个逻辑结果来实现多条件的联合判断。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 2

--- 如果真 (变量1 > 1 且 变量1 < 10)  
↓ 信息框 (“变量值在1与10之间”, 0, )

这段程序功能为：判断“变量1”的值是否在1与10之间。如果是，则显示提示信息。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 10

--- 如果真 (变量1 < 3 或 变量1 > 8)  
↓ 信息框 (“变量值不在3与8之间”, 0, )

这段程序的功能为：判断“变量1”的值是否小于3或大于8。如果是，则显示提示信息。

### 3.2.3 循环类流程控制命令

循环类流程控制命令可以在一定条件下多次执行一段代码，以减轻编程者的劳动量。例如，从1累加到100。

**注解：**由于循环中的语句会被多次的执行，所以执行循环块中的语句消耗系统资源也会随循环次数增长而增加，编程时应尽量优化循环块内的代码及减少不必要的循环次数，以减轻对资源的消耗。当循环块中的语句在循环全部结束前已经完成预期目标时，应使用跳转类流程控制命令跳过接下来没有有必要继续执行的语句，以节省资源。这样可以让编写出的程序执行速度快且占用资源少。

循环流程类命令都由循环首和循环尾两部分组成，它们成对出现，不会单一存在。在代码中输入了循环首命令，循环尾就自动出现。循环首表示循环的开始，循环尾表示循环的结束，循环首和循环尾之间的代码称为循环块，是循环类命令要重复执行的代码。

1) “判断循环首 ()”和“循环判断首 ()”命令

“判断循环首”命令

命令原型为：

<无返回值> 判断循环首 (逻辑型 条件)

“判断循环首 ()”命令首先检查判断条件是否成立。如果不成立，直接跳到循环尾后的代码继续执行；如果条件成立，则进入循环。每次循环结束后，会再一次检查“判断循环首”中的条件，如果条件不成立了，则退出循环，执行后续代码。





例如：输出100以内的偶数，代码如下所示：

变量名	类型	静态	数组	备注
变量	整数型			

```
变量 = 0
--> 判断循环首 (变量 < 100)
    变量 = 变量 + 2
--> 输出调试文本 (变量)
--- 判断循环尾 ()
```

“循环判断首”命令

命令原型为：

<无返回值> 循环判断首 ()

“循环判断首 ()”命令是先循环再判断，即首先执行一次循环块，再判断条件是否成立。如果“循环判断尾 ()”中的条件为“真”，就跳到循环首处继续新一次循环，如果条件不成立，则循环终止，向下执行其他代码。“判断循环首”、“循环判断首”这两个命令在一定情况下是可以互换的，但由于两个命令的判断顺序不同，有可能对循环体内的运行结果造成影响，在实际应用中要注意区分。

例如：输出100以内的奇数，代码如下所示：

变量名	类型	静态	数组	备注
变量	整数型			

```
变量 = 1
--> 循环判断首 ()
--> 输出调试文本 (变量)
    变量 = 变量 + 2
--- 循环判断尾 (变量 < 100)
```

2) “计次循环首 ()”命令

命令原型为：

<无返回值> 计次循环首 (整数型 循环次数, [整数型变量 已循环次数记录变量])

“计次循环首 ()”命令可以指定循环的次数。命令的第二个参数可以填入一个变量，用来记录已循环次数，第一次循环时变量值为1，第二次循环时变量值为2，依此类推。

下面编写一个输出1~100的所有整数和的程序，代码如下所示：

变量名	类型	静态	数组	备注
累加变量	整数型			
计次变量	整数型			

```
--> 计次循环首 (100, 计次变量)
    累加变量 = 累加变量 + 计次变量
--- 计次循环尾 ()
--> 输出调试文本 (累加变量)
```





代码说明：

在计次循环首内部每次循环让“累加变量”的数值等于它当前的数值加上循环的次数（这里的循环次数从1开始到100结束）。

### 3) “变量循环首 ()” 命令

命令原型为：

<无返回值> 变量循环首 (整数型 变量起始值, 整数型 变量目标值, 整数型 变量递增值, [整数型变量 循环变量])

该命令用做一个变量的内部循环，有四个参数，规定了变量的起始值，目标值和递增值，每次循环，变量的起始值都会增加指定的递增值，直到达到或超过目标值，退出循环。第四个参数可以传递一个变量，来接收循环时变量的当前值。

例如：可以求出100~200之间的整数和，代码如下所示：

变量名	类型	静态	数组	备注
变量1	整数型			
相加变量	整数型			

--> 变量循环首 (100, 200, 1, 变量1)

相加变量 = 相加变量 + 变量1

--- 变量循环尾 ()

→ 输出调试文本 (相加变量)

循环结束后的“相加变量”就是求得的结果。最后将结果显示在输出面板中。

### 4) 循环的嵌套

在易语言中所有循环类命令都是可以嵌套使用的。

**【例】** 对成员数为10的数组进行排序。

分析：将数组内的成员两两比较它们的值，如果它们的值不满足顺序，则交换它们的值。直到所有成员都按照顺序排列为止。

首先我们定义一个成员数为10的数组，并将每个成员取一个1到100之间的随机数：

变量名	类型	静态	数组	备注
数据组	整数型		10	
循环变量1	整数型			
循环变量2	整数型			
临时变量	整数型			

给每个成员随机取一个值

--> 计次循环首 (10, 循环变量1)

置随机数种子 ()

数据组 [循环变量1] = 取随机数 (1, 100)

--- 计次循环尾 ()





将这个数组按照成员值从小到大排列：

第一层循环的主要作用是使“循环变量1”由1增长到10。“循环变量1”将作为数组的下标在后面使用，所以相当于将数组中所有成员遍历。

第二层循环的主要作用是使“循环变量2”由“循环变量1”+1起增长到10。这样做的目的是为了从“循环变量1”作为下标的后一个成员开始遍历它后面的成员。

然后判断后面成员的值是否小于当前（“循环变量1”为下标）的成员值；如果条件成立，那么交换它们的值。

```

' 将数组中成员按照从小到大排序
--> 计次循环首 (10, 循环变量1)
    --> 变量循环首 (循环变量1 + 1, 10, 1, 循环变量2)
        -- 如果真 (数据组 [循环变量2] < 数据组 [循环变量1])
            临时变量 = 数据组 [循环变量2]
            数据组 [循环变量2] = 数据组 [循环变量1]
            数据组 [循环变量1] = 临时变量
        -- 变量循环尾 0
    -- 计次循环尾 0

```

最后输出一下排序后的结果：

```

' 输出每个成员的值
--> 计次循环首 (10, 循环变量1)
    输出调试文本 (数据组 [循环变量1])
-- 计次循环尾 0

```

为了便于理解，假设一组数据，来了解一下排序的过程。

假设随机获得的成员数据为：80、70、9、67、11、100、40、85、3、39。

进入第一层循环，第一层循环控制了“循环变量1”从1开始，这时为1。

进入第二层循环，第二层循环控制了“循环变量2”从“循环变量1”的值+1开始，这时为2。

判断“数据组[2]”是否小于“数据组[1]”，即70是否小于80。

条件满足，进入满足条件所执行的代码块。

将“数据组[2]”当前的值（即70）保存在“临时变量”中；

将“数据组[1]”的当前值（即80）保存在“数据组[2]”中。

再将“临时变量的值”保存到“数据组[1]”中，到此就完成了两个成员值的交换。

第二层循环第一遍执行完成，跳到第二层循环首继续执行。

这时数组各成员值为：

70、80、9、67、11、100、40、85、3、39。





第二层循环第二遍执行，“循环变量2”的值自动增长1，这时为3。

判断“数据组[3]”是否小于“数据组[1]”

.....  
.....

当第二层循环全部执行完毕后，数组中的第一个成员也就是整个数组中值最小的成员。

即第一个成员完成了排序，固定了它的位置。

进入第一层循环的第二次执行，重复上面的步骤，直到整个数组中的成员全部排序完成。

结果为：

3、9、11、39、40、67、70、80、85、100。

本例程源码见随书光盘中“\图书例程\第三章\流程控制\排序数组.e”

### 3.2.4 跳转类流程控制命令

有了跳转类流程控制命令，可以更加方便的控制程序的流程。“跳出循环（）”和“到循环尾（）”命令可以控制循环的执行流程，“返回（）”和“结束（）”命令可以控制子程序和整个程序的执行流程。

1) “到循环尾（）”和“跳出循环（）”命令

命令原型为：

<无返回值> 到循环尾（）

<无返回值> 跳出循环（）

这两个跳转类流程控制命令都是用来控制循环的。当一个循环中运行了“到循环尾（）”命令，将会直接跳到循环尾的代码处；当一个循环中运行了“跳出循环（）”命令，那么当前的循环就会结束，然后继续运行循环体以后的代码。

(1) 让一个编辑框中只显示10以内的奇数“1、3、5、7、9”，代码如下所示：

变量名	类型	静态	数组	备注
循环变量	整数型			

```

--> 计次循环首 (9, 循环变量)
    -- 如果真 (循环变量 % 2 = 0)
        到循环尾 ()
    -- 编辑框1.加入文本 (到文本 (循环变量) + " ")
--> 计次循环尾 ()
  
```

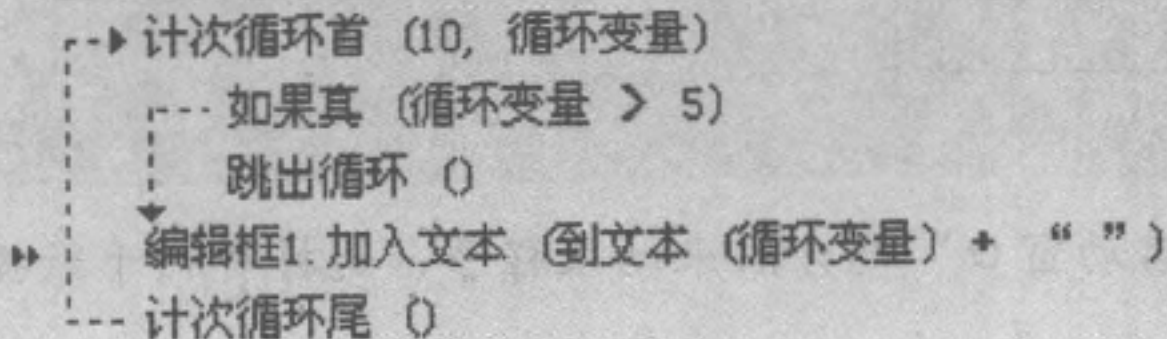
当循环变量为偶数时，就会执行“到循环尾（）”命令，跳过显示到编辑框的代码。

(2) 如果在循环过程中，当某个条件产生，想提前结束该循环，可以使用“跳出循环（）”命令。





变量名	类型	静态	数组	备注
循环变量	整数型			



当循环变量等于6，就满足了“如果真”的判断，将执行“跳出循环（）”命令，循环即会结束，所以编辑框只会显示“12345”。

### 2) “返回（）”命令

命令原型为：

<无返回值> 返回（[通用型 返回到调用方的值]）

“返回（）”命令被执行后，会退出当前子程序。当前子程序中“返回（）”命令之后的代码将不再被执行，程序将自动跳转到调用本子程序语句的下一条语句处继续执行。

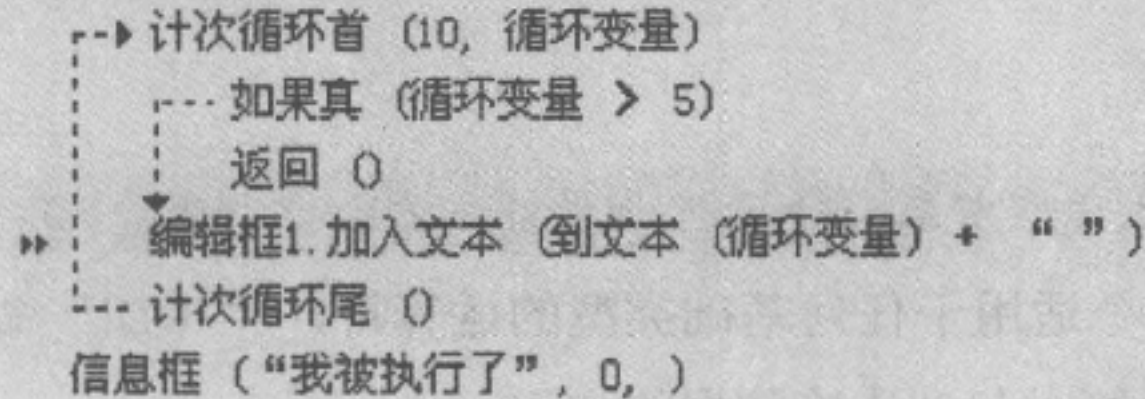
因为“返回（）”命令后面的代码不被运行，所以“返回（）”命令也经常被用于程序的调试。有返回值的子程序必须使用“返回（）”命令来返回执行结果，要注意实际返回值的数据类型要和子程序定义的返回值数据类型相匹配。

例如：返回日期时间型的数据

返回（[2009年1月19日]）

在循环中使用“返回（）”命令，也可以终止一个循环的运行。

变量名	类型	静态	数组	备注
循环变量	整数型			



当循环变量大于5时，满足“如果真”的判断条件，执行了“返回（）”命令使当前循环中和当前子程序中的后续代码都不会被执行。

### 3) “结束（）”命令

命令原型为：

<无返回值> 结束（）

“结束（）”命令是结束当前易程序的运行。可以实现程序的关闭功能。“结束（）”命令没有返回值也没有参数。若仅想关闭当前的某一个窗口，应使用“窗口”的“销毁（）”命令。





例如：在按下“关闭按钮”后结束程序：

子程序名	返回值类型	公开	备注
_关闭按钮_被单击			

结束 ()

**注解：**尽可能在程序中使用销毁“\_启动窗口”的方法来结束程序，这样有利于数据的安全。

## 3.3 算术运算命令

我们将把算术运算部分分为两部分进行讲解。

### (1) 基本算术运算命令及其运算符

基本算术运算命令在程序中最常使用的命令之一，几乎每个程序都离不开它。

### (2) 扩展算术运算命令

扩展算术运算命令是易语言中为了方便编程者而提供的。它们是一些与数学运算相关的命令。

### 3.3.1 基本算术运算命令及其运算符

基本运算命令有：相加（+）、相减或负（-）、相乘（\*）、相除（/）、整除（\）、求余数（%）。

#### 1) “相加”命令。

“相加”命令：简化运算符“+”。

命令原型为：

<通用型> 相加（通用型 被加数或文本或字节集，通用型 加数或文本或字节集，...）

相加（+）命令是基本运算符中唯一一个适用于任何基础类型的运算符，“相加”命令在数字运算时用于数字相加；在非数字运算时起到连接数据的作用。

当在两个数字之间使用“相加”命令则是将它们以数学方式相加，并返回之和。

如：5+6返回的是11。

又如：“ABC”+“DEF”，这个表达式是将ABC与DEF这两个文本连接，而不是将它们以数学方式相加。

**注解：**使用“相加”命令时需要注意，数字与数字相加时，在易语言内部是以双精度小数型进行计算的。

#### 2) “相减”命令与“负”命令：简化运算符“-”

命令原型为：





<双精度小数型> 相减 (双精度小数型 被减数, 双精度小数型 减数, ...)

<双精度小数型> 负 (双精度小数型 数值)

相减 (-) 命令会将两个数值进行求差运算, 而“负”命令则是改变一个数的符号。

“相减”命令与“负”命令的运算符同为“-”, 其实只是使用在不同位置时的名称不同而已, 它们的概念可以互相转换。

例如:  $5-5$ 与 $5+(-5)$ 从数学理论上可以等同的看待, 在易语言中也同样。又如在易语言中输入可以简化的数学式后, 在代码预编译时会自动将其简化。

3) “相乘”命令: 简化运算符“\*”

命令原型为:

<双精度小数型> 相乘 (双精度小数型 被乘数, 双精度小数型 乘数, ...)

相乘 (\*) 命令会将两个数进行求积运算。

例如:  $4*5$ 返回的是20。

4) “相除”命令: 简化运算符“/”

命令原型为:

<双精度小数型> 相除 (双精度小数型 被除数, 双精度小数型 除数, ...)

相除 (/) 命令会将两个数进行求商运算。

例如:  $30 / 6$ 返回的是5。

**注意:** 除数不能为0, 因为数学中除数为0没有意义, 因此易语言程序中如果除数为0将会引发程序错误。

5) “整除”命令: 简化运算符“\”

命令原型为:

<双精度小数型> 整除 (双精度小数型 被除数, 双精度小数型 除数, ...)

整除 (\) 命令会将两个数进行求商运算并舍弃小数部分。

例如:  $21 \backslash 4$ 返回的是5。

**注意:** 除数不能为0, 原因同“相除”命令。

6) “求余数”命令: (%)

命令原型为:

<双精度小数型> 求余数 (双精度小数型 被除数, 双精度小数型 除数, ...)

求余数 (%) 命令会计算出被除数与除数求商后的余数。

例如:  $21 \% 4$ 返回的是1

**注意:** 除数不能为0, 原因同“相除”命令。关于基本算术运算命令的优先级可参看本书第2.2节。





### 3.3.2 扩展算术运算命令

#### 1) “取符号”命令

命令原型为:

<整数型> 取符号 (双精度小数型 欲取其符号的数值)

本命令用于获取数值或数值型变量的正负符号, 如果为正数则返回大于零; 如果为负数则返回小于零; 为0时返回0。

例如:

输出调试文本(取符号(-411.14))

运行后输出: -1。

#### 2) “取绝对值”命令

命令原型为:

<双精度小数型> 取绝对值 (双精度小数型 欲取其绝对值的数值)

本命令用于获取数值或数值型变量的绝对值。

例如:

输出调试文本(取绝对值(-3.14))

运行后输出: 3.14。

#### 3) “取整”命令

命令原型为:

<整数型> 取整 (双精度小数型 欲取整的小数)

本命令用于获取数值或数值型变量的整数部分。但“取整”命令在对正小数和负小数的处理时有不同的效果。

当对一个负小数取整时, 如果有小数部分, 则结果为原整数部分减1。

例如: 对-5.1取整。

输出调试文本(取整(-5.1))

获得的结果为-6。

当对一个正小数取整时, 小数部分将会完全被抛弃。

例如: 对4.9取整。

输出调试文本(取整(4.9))

获得的结果为4。

#### 4) “绝对取整”命令

命令原型为:

<整数型> 绝对取整 (双精度小数型 欲取整的小数)



本命令用于获取一个数值或数值型变量的整数部分。“绝对取整”命令与“取整”命令不同之处在于，不论被取整的数是正数、负数，小数部分将被完全抛弃。

**【例】** 对-5.1绝对取整。

输出调试文本(绝对取整(-5.1))

获得的结果为-5。

**【例】** 对4.9绝对取整。

输出调试文本(绝对取整(4.9))

获得的结果为4。

#### 5) “四舍五入”命令

命令原型为：

<双精度小数型> 四舍五入 (双精度小数型 欲被四舍五入的数值, [整数型 被舍入的位置])

本命令用于获取一个数值或数值型变量的四舍五入值。被舍弃的位置通过它的参数指定。

参数<2>的设置起到指定舍弃位置的作用。当它是0时，意味着不保留小数，对十分位进行四舍五入；当它是大于0时，表示保留的小数位数，它的下一位小数将进行四舍五入；当它是小于0时，表示对整数部分的从某一位开始四舍五入，舍弃此参数的绝对值减1的整数位数（用0占位），小数部分将被抛弃。

**【例】** 对1583.456四舍五入，第二个参数设为-2（从小数点开始，向左2位）。

输出调试文本(四舍五入(1583.456, -2))

得到的结果为1600。

#### 6) “求次方”命令

命令原型为：

<双精度小数型> 求次方 (双精度小数型 欲求次方数值, 双精度小数型 次方数)

本命令用于获取一个数值或数值型变量的N次方。与数学一样，也可以使用分指数的概念使用本命令来开一个数的N次方。

**【例】** 求次方(4, 0.5)即为开4的2次方。

输出调试文本(求次方(4, 0.5))

得到的结果为2。

#### 7) “求平方根”命令

命令原型为：

<双精度小数型> 求平方根 (双精度小数型 欲求其平方根的数值)

本命令用于获取一个数值或数值型变量的平方根。

例如：求25的平方根。

输出调试文本(求平方根(25))





运行后输出：5。

#### 8) “求正弦”命令

命令原型为：

<双精度小数型> 求正弦 (双精度小数型 欲进行计算的角)

本命令用于获取一个弧度的正弦值。因为使用的单位是弧度值，所以如果需要计算一个角度的正弦值时应按照数学方法转换。即将角度值乘以 $\pi/180$ ，在易语言中 $\pi$ 为一个常量。常量名为#pi。

例如：求正弦命令取 $90^\circ$ 角的正弦值。

输出调试文本(求正弦( $90 \times \#pi \div 180$ ))

结果为1。

**注解：**在易语言中#pi的值被固定为3.1415926535，当在进行精密计算时要注意误差精度。

#### 9) “求余弦”命令

命令原型为：

<双精度小数型> 求余弦 (双精度小数型 欲进行计算的角)

本命令用于获取一个弧度的余弦值。

#### 10) “求正切”命令

命令原型为：

<双精度小数型> 求正切 (双精度小数型 欲进行计算的角)

本命令用于获取一个弧度的正切值。

#### 11) “求反正切”命令

命令原型为：

<双精度小数型> 求反正切 (双精度小数型 欲求其反正切值的数值)

本命令用于通过一个正切值求出它所对应的弧度值。若要将弧度转换为角度，应按照数学中的方法将弧度值乘以180再除以 $\pi$ 。在易语言中 $\pi$ 一般使用#pi常量代替。

#### 12) “求自然对数”命令

命令原型为：

<双精度小数型> 求自然对数 (双精度小数型 欲求其自然对数的数值)

本命令用于取以e为底的对数。e在易语言中使用#e常量代替。#e在易语言中的固定值为2.718282，当在进行精密计算时要注意误差精度。

#### 13) “求反对数”命令

命令原型为：

<双精度小数型> 求反对数 (双精度小数型 欲求其反对数的数值)

本命令用于取以e（自然对数的底）为底的某次方。在使用时应注意数据类型的范





围，避免溢出。

#### 14) “是否运算正确”命令

命令原型为：

<逻辑型> 是否运算正确 (双精度小数型 欲校验的计算结果)

本命令用于检查算术运算命令的返回结果是否是正常的。如果该数值正确有效，返回真；如果该数值是运算错误或运算溢出后的结果，则返回假。

检查结果的有效范围为：

“乘”、“除”、“求次方”、“求平方根”、“求正弦值”、“求余弦值”、“求正切值”、“求反正切值”、“求自然对数”、“求反对数”。

【例】求平方根-1是否运算正确。

输出调试文本(是否运算正确(求平方根(-1)))

在输出面板中输出假，表示运算结果不正确。

#### 15) “置随机数种子”命令

命令原型为：

<无返回值> 置随机数种子 ([整数型 欲置入的种子数值])

本命令用于设置产生随机数的基数。如果调用时不给参数，默认是使用当前系统启动时间到当前时间的毫秒数作为基数。在取随机数之前调用此命令可在一定程度上避免取得的随机数相同或有规律。

#### 16) “取随机数”命令

命令原型为：

<整数型> 取随机数 ([整数型 欲取随机数的最小值], [整数型 欲取随机数的最大值])

本命令用于在指定范围中获取一个伪随机数。

例如：取1~100的随机数。

置随机数种子 0

输出调试文本(取随机数(1, 100))

在输出面板中输出一个1~100的随机数。

## 3.4 逻辑比较

在易语言中逻辑比较类命令只能比较基本数据类型。即：字节型、短整数型、整数型、长整数型、小数型、双精度小数型、子程序指针、逻辑型、文本型、字节集型、日期时间型。



### 3.4.1 逻辑比较命令

1) “等于”命令：运算符号为“=”或“==”

命令原型为：

<逻辑型> 等于 (通用型 被比较值, 通用型 比较值)

本命令用于比较两个数据是否一致。如果被比较值与比较值一致，则返回“真”；否则返回“假”。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。

双精度小数型与小数型无须转换就可以直接进行比较。

2) “不等于”命令：运算符号为“<>”或“!=”或“≠”

命令原型为：

<逻辑型> 不等于 (通用型 被比较值, 通用型 比较值)

本命令用于比较两个数据是否不一致。如果被比较值与比较值不一致，则返回“真”；否则返回“假”。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。

双精度小数型与小数型无须转换就可以直接进行比较。

3) “小于”命令：运算符号为“<”

命令原型为：

<逻辑型> 小于 (通用型 被比较值, 通用型 比较值)

本命令用于比较一个数据是否小于另一个数据。如果是则返回“真”；否则返回“假”。当本命令用来比较两个文本时，按照一一对应比较文本中每个字符的ASCII码的值。当比较两个日期时间型数据时，比较顺序为年、月、日、时、分、秒，各项以数值大小做比较。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。

双精度小数型与小数型无须转换就可以直接进行比较。

4) “大于”命令：运算符号为“>”

命令原型为：

<逻辑型> 大于 (通用型 被比较值, 通用型 比较值)

本命令用于比较一个数据是否大于另一个数据。如果是则返回“真”；否则返回“假”。当本命令用来比较两个文本时，按照一一对应比较文本中每个字符的ASCII码的值。当比较两个日期时间型数据时，比较顺序为年、月、日、时、分、秒，各项以数值大小做比较。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。





双精度小数型与小数型无须转换就可以直接进行比较。

5) “小于或等于”命令：运算符号为“<=”或“≤”

命令原型为：

<逻辑型> 小于或等于 (通用型 被比较值, 通用型 比较值)

本命令用于比较一个数据是否小于或者等于另一个数据。如果是则返回“真”；否则返回“假”。当本命令用来比较两个文本时，按照一一对应比较文本中每个字符的ASCII码的值。当比较两个日期时间型数据时，比较顺序为年、月、日、时、分、秒，各项以数值大小做比较。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。

双精度小数型与小数型无须转换就可以直接进行比较。

6) “大于等于”命令：运算符号为“>=”或“≥”

命令原型为：

<逻辑型> 大于或等于 (通用型 被比较值, 通用型 比较值)

本命令用于比较一个数据是否大于或者等于另一个数据。如果是则返回“真”；否则返回“假”。当本命令用来比较两个文本时，按照一一对应比较文本中每个字符的ASCII码的值。当比较两个日期时间型数据时，比较顺序为年、月、日、时、分、秒，各项以数值大小做比较。

**注解：**比较值的数据类型必须与“被比较值”一致或者可以相互转换。

双精度小数型与小数型无须转换就可以直接进行比较。

7) “近似等于”命令：运算符号为“?=”或“≈”

命令原型为：

<逻辑型> 近似等于 (文本型 被比较文本, 文本型 比较文本)

本命令用于判断一个文本内是否包含另一个文本。如果是则返回“真”；否则返回“假”。

需要注意的是：比较时是从两个文本的第一个字符开始一一对应比较的。如果用作比较的文本是在被比较的文本中的一段，本命令将依然返回“假”。

例如：

变量名	类型	静态	数组	备注
b	逻辑型			

b = “abcd” ≈ “bcd”

执行后变量b中的值为“假”。

8) “并且”命令：运算符号为“&&”或“And”或“且”

命令原型为：

<逻辑型> 并且 (逻辑型 逻辑值一, 逻辑型 逻辑值二, ...)





本命令用于判断两个或几个逻辑值是否都为“真”。如果是则返回“真”；否则返回“假”。判断时如果其中一个参数逻辑值为“假”时，则命令立即返回，不会再判断下面的参数（若作为参数的是一个子程序或命令的调用，那么它将不会被执行到）。

9) “或者”命令：运算符为“||”或“Or”或“或”

命令原型为：

<逻辑型> 或者 (逻辑型 逻辑值一, 逻辑型 逻辑值二, ...)

本命令用于判断两个或几个逻辑值中是否有一个为“真”。如果是则返回“真”；否则返回“假”。判断时如果其中一个参数逻辑值为“真”时，则命令立即返回，不会再判断下面的参数（若作为参数的是一个子程序或命令的调用，那么它将不会被执行到）。

10) “取反”命令

命令原型为：

<逻辑型> 取反 (逻辑型 被反转的逻辑值)

本命令用于获取一个逻辑值或逻辑型变量的值的相反值。既如果被取反的值为“真”，则本命令返回“假”；如果为“假”则返回“真”。

### 3.4.2 多条件逻辑比较时的运算顺序

当一条逻辑判断条件中既使用“或者”命令又使用了“并且”命令时，易语言中的运算顺序为：从左向右依次执行，并使用“或者”命令将多个条件分割，然后将使用“并且”命令连接的判断返回值视为一个条件。最终的整个判断的返回值是由“或者”命令返回的。也可以理解为先处理“并且”，再处理“或者”，但要注意程序并不是预先处理所有“并且”再处理“或者”。

例如：

变量名	类型	静态	数组	备注
a	整数型			
b	整数型			
c	整数型			

a = 1

b = 2

c = 3

输出调试文本 (b > 2 或 a < c 且 a < b 或 a > b)

上面这段程序的执行效果为在输出面板中显示b > 2 或 a < c 且 a < b 或 a > b的比较结果。显示内容为“真”。

程序执行顺序为：

判断b的值是否大于2，结果为“假”，继续判断。

判断a的值是否小于c并且a的值是否小于b，结果为“真”，结束判断，返回结果（“真”）。





在多条件逻辑比较时同样可以像使用数学运算符时一样，使用小括号（）来强行改变判断顺序。

例如：

变量名	类型	静态	数组	备注
a	整数型			
b	整数型			
c	整数型			

```
a = 1  
b = 2  
c = 3  
» 输出调试文本 ((b > 2 或 a < c) 且 (a < b 或 a > b))
```

上面这段程序的执行效果为在输出面板中显示（b > 2 或 a < c）且（a < b 或 a > b）的比较结果。显示内容为“真”。

程序执行顺序为：

判断b的值是否大于2，结果为“假”，继续判断；

判断a的值是否小于c，结果为“真”，继续判断；

判断a的值是否小于b，结果为“真”，继续判断。

判断“且”的左右两个条件是否都为“真”，结果为“真”，结束判断，返回结果。

虽然这个例子和上一个例子的返回结果相同，但它们的意义及执行顺序却完全不同，所以在理解时不要混淆两者。

## 3.5 位运算命令

### 3.5.1 了解位运算

位运算是指对数据进行二进制的逐位运算。我们知道现代电子计算机内部是采用二进制方式存储和处理数据的，输入到计算机的数字、字母、汉字等信息都以二进制的形式存储。在计算机内部用“比特（bit）”这个单位来表示一个二进制，其状态只有0或1两种，位运算命令就是直接改变这些“比特”来达到改变数据的效果。要想掌握位运算命令，必须了解二进制。所以我们下面先从二进制开始讲起。

### 3.5.2 常用进制

在计算机内部运算中常用的进位制有4种。

二进制：逢2进1，由数字0和1组成，以下标2或后缀B表示。







八进制：逢8进1，由数字0至7组成，以下标8或后缀Q表示。

十进制：逢10进1，由数字0至9组成，以下标10后缀D表示，该后缀可以省略。

十六进制：逢16进1，由数字0至9和字母A至F组成，以下标16或后缀H表示。

例如：2进制数1001010表示为1001010 (B)、八进制数234512表示为234512 (Q)、十六进制数4523ADF表示为4523ADF (H)，十进制数的后缀可以省略。

所谓二进制，就是以“逢二进一，借一当二”为原则。对数值进行计数的进位制，和我们日常使用的十进制类似，只不过十进制是“逢十进一，借一当十”。

二进制数系统中，位简记为b,也称为比特，每个0或1就是一个位 (bit)，位是电子计算机中数据存储的最小单位。

### 1) 十进制数和二进制数之间的转换

(1) 二进制数转换为十进制数。二进制数各位乘以与其对应的“权”之和即为该二进制数对应的十进制数。“权”是指以 $2^n$ 代表二进制数的位。如：

$$1011100.10111B = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} = 92.71875 (D)$$

(2) 十进制数转换为二进制数。十进制数转换为其他进制数时，整数部分和小数部分分别进行转换。整数部分采用除基取余法，小数部分采用乘基取整法。

### 2) 使用除基取余法转换整数的方法

(1) 如果十进制整数小于要转换成的进位制的基数，则此十进制整数就是要转换成的进位制表示的数。

如：将十进制整数6转换成八进制数，由于 $6 < 8$ ，十进制整数6也可以看作八进制数6，即 $6(D) = 6(Q)$ 。

(2) 如果十进制整数大于要转换成的进位制的基数，用该十进制整数除以要转换成的进位制的基数，取余数，该余数就是转换后的低位数。此时，如果所得的商小于基数，则商就是转换后的高位数。

例如：十进制数12转换成八进制数， $12/8$ ，商为1，余数为4，由于 $4 < 8$ ，不再继续转换，则 $12(D) = 14(Q)$ 。

(3) 如果上一步所得的商仍然大于要转换成的进位制的基数，继续第二步的操作，直到商小于基数为止。所取得的余数和商按下列顺序从左到右排列：

商、最后得到的余数、…、第二个得到的余数、第一个得到的余数。

如：将十进制数205转换成二进制数，过程如下：

205/2，商为102，第一个余数1；

102/2，商为51，第二个余数为0；

51/2，商为25，第三个余数为1；

25/2，商为12，第四个余数为1；





12/2, 商为6, 第五个余数为0;

6/2, 商为3, 第六个余数为0;

3/2, 商为1, 第七个余数为1。

此时, 商1小于基数2, 既转换完毕, 将得到的余数按从后向前排列则得到的二进制数为:

11001101 (最终商、第七个余数、第六个余数、第五个余数、第四个余数、第三个余数、第二个余数、第一个余数)。

带小数数值的转换方法为: 将整数部分按除基法运算, 将小数部分按乘基法运算, 再将两部分结果相加。

使用乘基取整法转换小数的方法是:

对于被转换的十进制数的小数部分则应不断的乘以基数, 并记下其整数部分, 再取出小数部分重复刚才的步骤, 直到结果的小数部分为0为止。

例如: 将十进制数0.6875转换为二进制数, 过程如下:

$0.6875 \times 2 = 1.375$ , 小数部分第一位为1;

$0.375 \times 2 = 0.75$ , 小数部分第二位为0;

$0.75 \times 2 = 1.5$ , 小数部分第三位为1;

$0.5 \times 2 = 1.0$ , 小数部分第四位为1。

由于最后一步所得的乘积的小数部分为0, 转换结束, 结果为0.1011。

### 3) 十六进制数、八进制数与二进制数之间的转换

一位十六进制数用四位二进制数表示, 一位八进制数用三位二进制数表示。

二进制数转换为十六进制数时, 以小数点位置为界, 向两侧每四位分组, 当两侧不足四位时补0。

例如:  $101010.010101 (B) = 0010\ 1010.0101\ 0100 (B) = 2A.54 (H)$

二进制数转换为八进制数时, 以小数点位置为界, 向两侧每三位分组, 当两侧不足三位时补0。

例如:  $101010.010101 (B) = 101, 010.010, 101 (B) = 52.25 (Q)$

十六进制数转换为二进制数时, 以小数点为界, 每一位十六进制数转换为四位二进制数向两侧排列; 八进制数转换为二进制数时, 以小数点为界, 每一位八进制数转换为三位二进制数向两侧排列。

## 3.5.3 位运算命令

在易语言中, 所有位运算命令都是针对“整数型”数据进行操作的, 而“整数型”长度固定为4个字节=32比特(位), 也就是一组32位长度的二进制数。

### 1) “位取反”命令

命令原型为:



<整数型> 位取反 (整数型 欲取反的数值)

“位取反 ( )”命令用于对一个数据的二进制值的每一位取反, 即0变为1, 1变为0, 返回值是转换后的十进制数。

例如：

编辑框\_位取反.内容 = 到文本(位取反(80))

代码运行后，编辑框会显示“位取反”运算结果“-81”。

内部运算过程为：

将80转换为二进制值，即0000000000000000000000000101000。

将每位取反，即1111111111111111111111101011。

将取反结果转换为整数型（十进制），即-81。

## 2) “位与”命令

命令原型为:

<整数型> 位与 (整数型 位运算数值一, 整数型 位运算数值二, ...)

“位与（）”命令用于将2个或多个数据的二进制值中共同比特位进行“与”运算。即如两个或多个整数的共同位均为1，则返回值的对应位也为1，否则为0。运算完毕后，返回值是转换后的十进制数。例如：

一个二进制数的第四位为1，另一个二进制数的第四位为1，则返回值的第四位为1；

一个二进制数的第四位为0，另一个二进制数的第四位为1，则返回值的第四位为0；

一个二进制数的第四位为1，另一个二进制数的第四位为0，则返回值的第四位为0；

一个二进制数的第四位为0，另一个二进制数的第四位为0，则返回值的第四位为0。

例如：

编辑框\_位与.内容 = 到文本 (位与 (56, 89))

运行后可以得出的结果为“24”。

56和89分别转换成二进制数为：0011 1000和0101 1001，进行与的运算后即会得出结果“0001 1000”即“24”。

### 3) “位或”命令

命令原型为:

<整数型> 位或 (整数型 位运算数值一, 整数型 位运算数值二, ...)

“位或（）”命令对二进制数值进行“或”运算，并将运算后结果以十进制返回。如2个或多个数值的共同位均为0，则返回值的对应位也为0，否则为1。运算完毕后，返回值是转换后的十进制数。

一个数值的第四位为1，另一个数值的第四位为1，则返回值的第四位为1；

一个数值的第四位为0，另一个数值的第四位为1，则返回值的第四位为1；



一个数值的第四位为1，另一个数值的第四位为0，则返回值的第四位为1；

一个数值的第四位为0，另一个数值的第四位为0，则返回值的第四位为0。

例如：

编辑框\_位或.内容 = 到文本(位或(56, 89))

运行后的结果为“121”。

56和89分别转换成二进制数为：0011 1000和0101 1001，进行或的运算后即会得出结果“0111 1001”即“121”。

#### 4) “位异或”命令

命令原型为：

<整数型> 位异或 (整数型 位运算数值一, 整数型 位运算数值二, ...)

“位异或( )”命令对二进制数值的共同比特位进行“异或”运算，并将运算结果以十进制返回。如果两个或多个数值的共同位相等（均为0或均为1），则返回值的对应位就是0，否则为1。运算完毕后，返回值是转换后的十进制数。

例如：

一个数值的第四位为0，另一个数值的第四位为1，则返回值的第四位为1；

一个数值的第四位为1，另一个数值的第四位为0，则返回值的第四位为1；

一个数值的第四位为1，另一个数值的第四位为1，则返回值的第四位为0；

一个数值的第四位为0，另一个数值的第四位为0，则返回值的第四位为0。

例如：

编辑框\_位异或.内容 = 到文本(位异或(56, 89))

运行后的结果为“97”。

56和89分别转换成二进制数为：0011 1000和0101 1001，进行异或的运算后即会得出结果“0110 0001”。即“97”。

#### 5) “左移”命令

命令原型为：

<整数型> 左移 (整数型 欲移动的整数, 整数型 欲移动的位数)

“左移( )”命令是对二进制数据的每一位进行向左（高位）移动，移动后的空位自动用0补位。

参数<2>指定了欲移动的位数。如果该参数等于0或32的整数倍，则不进行移动操作并返回原值；如果该参数为正数，则先将该参数取32的余数，由右向左移动余数位；如果该参数为负数，则先将该参数取32的余数，并由右向左移动 32 + 余数位。

**【例】** 对5进行左移2位。

将5的二进制101（前29位为0占位，这里省略）的每一位向左移动2位，即10100（前27位为0占位，这里省略）。







再将结果以十进制返回。即20。

**【例】** 对5进行左移31位。

将5的二进制101（前29位为0占位，这里省略）的每一位向左移动31位，即101后31个0。但其超过了32位，所以最前面的10将被抛弃。最终结果为1后31个0。

再将结果以10进制返回。即-2147483648（易语言中整数型的最高一位用来表示是否为负数）。

6) “右移”命令

命令原型为：

**<整数型> 右移 (整数型 欲移动的整数, 整数型 欲移动的位数)**

“右移( )”命令是对二进制数据的每一位进行向右（低位）移动，移动后的空位自动用0补位。

参数<2>指定了欲移动的位数。如果该参数等于0或32的整数倍，则不进行移动操作并返回原值；如果该参数为正数，则先将该参数取32的余数，由左向右移动余数位；如果该参数为负数，则先将该参数取32的余数，并由左向右移动32+余数位。

**【例】** 对20进行右移2位。

将20的二进制10100（前27位为0占位，这里省略）的每一位向右移动2位，即101（前29位为0占位，这里省略）。

再将结果以十进制返回。即5。

**【例】** 对5进行右移2位。

将5的二进制101（前29位为0占位，这里省略）的每一位向右移动2位，即前31个0+101，但其超过了32位，所以最后面的01将被抛弃。最终结果为前31个0+1。

再将结果以10进制返回。即1。

7) “合并整数”命令

命令原型为：

**<整数型> 合并整数 (整数型 用作合并的整数1, 整数型 用作合并的整数2)**

“合并整数( )”命令是分别取两个整数的低16位并将他们组合成一个新的整数。从第一个参数中取得的16位将作为新整数的低16位；从第二个参数中取得的16位将作为新整数的高16位。

**【例】** 合并整数(2, 5)。

将2的二进制数据中的后（右）16位取出。即14个0+10；

将5的二进制数据中的后（右）16位取出。即13个0+101。

将由5中取得的作为高位，由2中取得的作为低位进行组合。组合后的二进制结果为13个0+101+14个0+10，并返回十进制结果。即327682。





### 8) “合并短整数”命令

命令原型为:

<短整数型> 合并短整数 (整数型 用作合并的整数1, 整数型 用作合并的整数2)

“合并短整数 ()”命令是分别取两个整数的低8位并将他们组合成一个新的短整数。从第一个参数中取得的8位将作为新整数的低8位; 从第二个参数中取得的16位将作为新整数的高16位。

**【例】** 合并短整数 (2, 5)。

将2的二进制数据中的后 (右) 8位取出。即0000 0010;

将5的二进制数据中的后 (右) 8位取出。即0000 0101。

将由5中取得的作为高位, 由2中取得的作为低位进行组合。组合后的二进制结果为0000 0101 0000 0010。并返回十进制结果。即1282。

## 3.6 数组操作命令

### 3.6.1 了解数组

数组: 即相同数据类型变量的一个集合。一个数组的所有变量拥有同一个数组名, 数组内的变量使用: 数组名[下标] 的方式表示数组内特定的一个变量, 对数组中的特定成员的使用与该类型变量相同。

在程序中使用数组不但方便管理相关的一组数据, 而且方便程序中使用循环控制一组数据。

扩展知识:

数组在内存中是一段连续的数据。数组下标从1开始, 数组可以定义为“多维”, 多维数组可以看作每个成员都是一个数组的数组。数组成员数和对应维数可以动态地进行更改。下面我们来学习易语言中的数组操作命令。

### 3.6.2 数组操作命令

#### 1) “重定义数组”命令

命令原型为:

<无返回值> 重定义数组 (通用型变量数组 欲重定义的数组变量, 逻辑型 是否保留以前的内容, 整数型 数组对应维的上限值, ...)

本命令用于在程序运行时动态地改变数组的结构。

参数<2>指定了重定义后是否保留以前的内容。但如果重定义后, 数组的对应维数小于原维数, 则超出现维数的成员将被抛弃。





例如：

变量名	类型	静态	数组	备注
数组1	整数型		5	

» 重定义数组 (数组1, 真, 4)

原数组是一个有5个成员的单维数组，执行重定义数组命令后，现数组有4个成员。原数组中的第5个成员将被抛弃。

## 2) “取数组成员数”命令

命令原型为：

<整数型> 取数组成员数 (通用型变量/变量数组 欲检查的变量)

本命令用于获取一个数组中的成员数量。当被检查的数组是多维数组时，返回值为所有维成员的总和。

例如：

变量名	类型	静态	数组	备注
数组1	整数型		5, 6	

» 输出调试文本 (取数组成员数 (数组1))

程序执行后，输出结果为30。

## 3) “取数组下标”命令

命令原型为：

<整数型> 取数组下标 (通用型变量/变量数组 欲取某维最大下标的数组变量，  
[整数型 欲取其最大下标的维] )

本命令用于获取数组指定维的成员数。

参数<2>指定了欲取数组中的第几维。如果给定的第一个参数不为数组变量或指定维不存在，则返回0。

例如：

变量名	类型	静态	数组	备注
数组1	整数型		5, 6	

» 输出调试文本 (取数组下标 (数组1, 2))

程序执行后，输出结果为6。

## 4) “复制数组”命令

命令原型为：

<无返回值> 复制数组 (通用型变量数组 复制到的数组变量，通用型数组 待复制的数组数据)

本命令用于创建一个现有数组结构及内容的拷贝。目标数组的原结构和数据将被完全抛弃。





例如：

变量名	类 型	静态	数组	备 注
数组1	整数型		5, 6	
数组2	整数型		0	

复制数组 (数组2, 数组1)  
» 输出调试文本 (取数组下标 (数组2, 2))

程序执行后，数组2的结构和数据与数组1完全相同。  
输出目前数组第二维的最大下标，结果为6。

5) “加入成员” 命令  
命令原型为：

<无返回值> 加入成员 (通用型变量数组 欲加入成员的数组变量, 通用型数组/非数组 欲加入的成员数据)

本命令用于向现有的数组中添加一个成员，添加的新成员在数组的最后。  
例如：

变量名	类 型	静态	数组	备 注
数组1	整数型		5	

加入成员 (数组1, 123)  
» 输出调试文本 (取数组下标 (数组1, 1))

程序执行后，在数组1的最后添加了一个新成员，新成员的值为123。  
输出目前数组第一维的最大下标，结果为6。

6) “插入成员” 命令  
命令原型为：

<无返回值> 插入成员 (通用型变量数组 欲插入成员的数组变量, 整数型 欲插入的位置, 通用型数组/非数组 欲插入的成员数据)

本命令用于向现有的数组中指定的位置插入一个成员。

参数<2>指定了欲插入成员的位置。位置值从 1 开始，如果小于 1 或大于现数组的成员数目 + 1，将不会插入任何数据。

例如：

变量名	类 型	静态	数组	备 注
数组1	整数型		5	

插入成员 (数组1, 3, 123)  
» 输出调试文本 (数组1 [3])

程序执行后，在数组1的第3个成员前插入了一个新成员，新成员的值为123。  
输出数组中的第3个成员的值，结果为123。





## 7) “删除成员”命令

命令原型为：

<整数型> 删除成员 (通用型变量数组 欲删除成员的数组变量, 整数型 欲删除的位置, [整数型 欲删除的成员数目])

本命令用于在现有数组中删除指定的成员。数组变量如为多维数组, 删除完毕后将转换为单维数组。命令执行后返回所实际删除的成员数目。

例如：

变量名	类型	静态	数组	备注
数组1	整数型		5, 6	

» 输出调试文本 (删除成员 (数组1, 10, 30))

» 输出调试文本 (取数组成员数 (数组1))

程序执行后, 数组1将被转换为单维数组。因为原数组共有30个成员, 从第10个 (包括第10个) 向后删除30个成员, 所以实际被删除的只有下标为10到30的成员。在输出面板中程序输出了实际被删除的成员数量 (21), 和数组现有成员的数量9。

## 8) “清除数组”命令

命令原型为：

<无返回值> 清除数组 (通用型变量数组 欲删除成员的数组变量)

本命令用于删除指定数组变量中的所有成员, 释放这些成员所占用的存储空间, 重新定义该变量为单维 0 成员数组变量。

例如：

变量名	类型	静态	数组	备注
数组1	整数型		5, 6	

清除数组 (数组1)

» 输出调试文本 (取数组成员数 (数组1))

程序执行后, 输出结果为0。

## 9) “数组排序”命令

<无返回值> 数组排序 (通用型变量数组 数值数组变量, [逻辑型 排序方向是否为从小到大])

本命令用于对一个数值型数组进行有序的排列。

参数<2>指定了排序的方式, 如果参数被省略则默认使用从小到大排序。如果被排序的数组是多维数组, 排序后的结果按照先低后高的维数进行排列。

例如: 排序一个二维数组, 两个维数下标上限都为2, 排序后的第一个结果将存放在[1][1]内, 第二个结果将存放在[1][2]内, 第三个结果将存放在[1][3]内, 其他多维数的数组依此类推。





例如：

变量名	类 型	静态	数组	备 注
数组1	整数型		2,2	

```
数组1 [1] [1] = 3  
数组1 [2] [1] = 2  
数组排序 (数组1, )  
» 输出调试文本 (数组1 [1] [1])
```

程序中首先对数组的[1][1]和[2][1]赋值，其次执行数组排序后数组将按照从小到大顺序排列，最后输出数组中[1][1]成员的值，即最小值。  
程序执行后，输出结果为0。

10) “数组清零” 命令  
命令原型为：

<无返回值> 数组清零 (通用型变量数组 数值数组变量)  
本命令用于对数值型数组的每个成员进行置0操作。  
例如：

变量名	类 型	静态	数组	备 注
数组1	整数型		2,2	

```
数组1 [1] [1] = 3  
数组清零 (数组1)  
» 输出调试文本 (数组1 [1] [1])
```

程序中首先对数组的[1][1]成员赋值，其次执行数组清零后数组中的每个成员的值都为0，最后输出[1][1]成员的值。  
程序执行后，输出结果为0。

**注解：**使用数组类命令时应当注意当使用重定义数组、复制数组、加入成员、插入成员、删除成员、清除数组命令后目标数组在内存中的地址将会改变。

### 3.7 环境存取命令

环境存取命令有以下几条。  
1) “取命令行” 命令  
命令原型为：

<无返回值> 取命令行 (文本型变量数组 存放被取回命令行文本的数组变量)





本命令可以取出在启动易程序时附加在其可执行文件名后面的所有以空格分隔的命令行文本段。本命令一般使用在程序具体功能的前边，用来处理所具体需要使用的功能或执行功能的附加参数。

为方便程序在易语言中调试，可以设置调试用的命令行。设置步骤如下：

易语言中“程序”菜单→“配置”→在“程序配置对话框”中的“通常”选项卡中的“调试参数行”处设置所需要的参数。多个参数使用空格分隔，如图3-3所示。

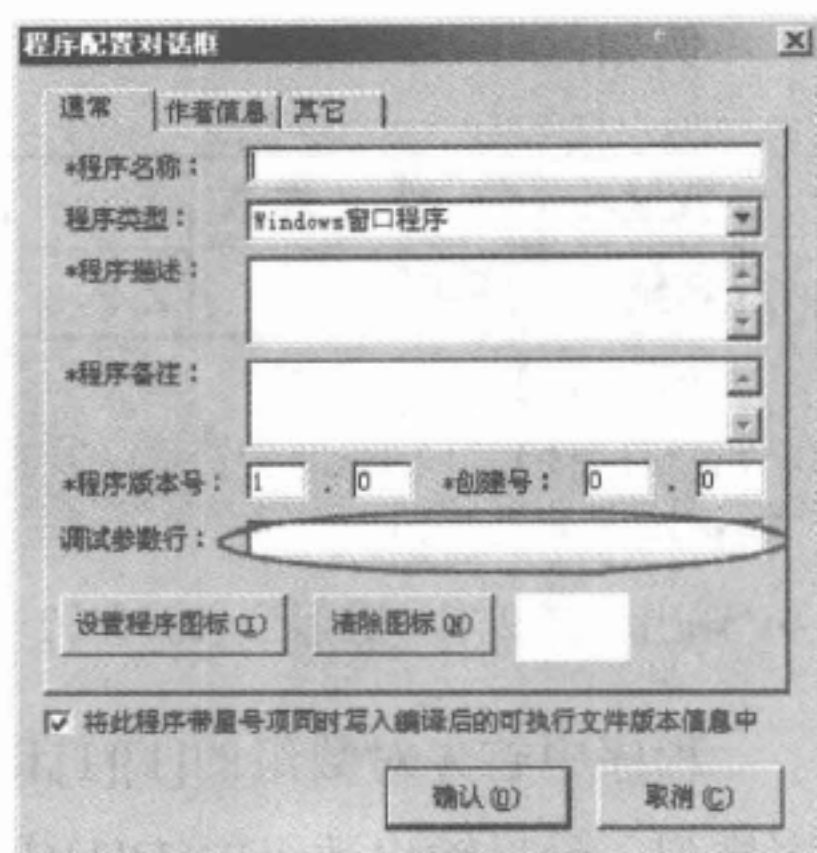


图3-3 调试用命令行设置

## 2) “取运行目录”命令

命令原型为：

<文本型> 取运行目录 ()

本命令用于获得所在程序运行时的可执行文件（exe）的所在路径。如果是在动态链接库（dll）中使用本命令，则返回的是调用动态链接库（dll）的可执行文件的所在路径。

例如：

输出调试文本(取运行目录())

程序执行后，在输出面板中显示程序所在的路径。

## 3) “取执行文件名”命令

命令原型为：

<文本型> 取执行文件名 ()

本命令用于获得调用本命令的可执行文件的文件名。如果是在动态链接库中使用本命令，则返回的是调用动态链接库的可执行文件的文件名。

例如：

输出调试文本(取执行文件名())

程序执行后，在输出面板中显示本程序的文件名（不包含路径）。

## 4) “读环境变量”命令

命令原型为：

<文本型> 读环境变量 (文本型 环境变量名称)

本命令用于在程序中获得Windows系统中的环境变量值。本命令的参数指明了欲获取的环境变量名称。欲查看系统中所有环境变量请在命令行中执行“set”命令。

执行方法为：开始→运行→Win 9x、ME输入command；WIN 2000以上输入cmd→确定→输入set→回车，如图3-4所示。





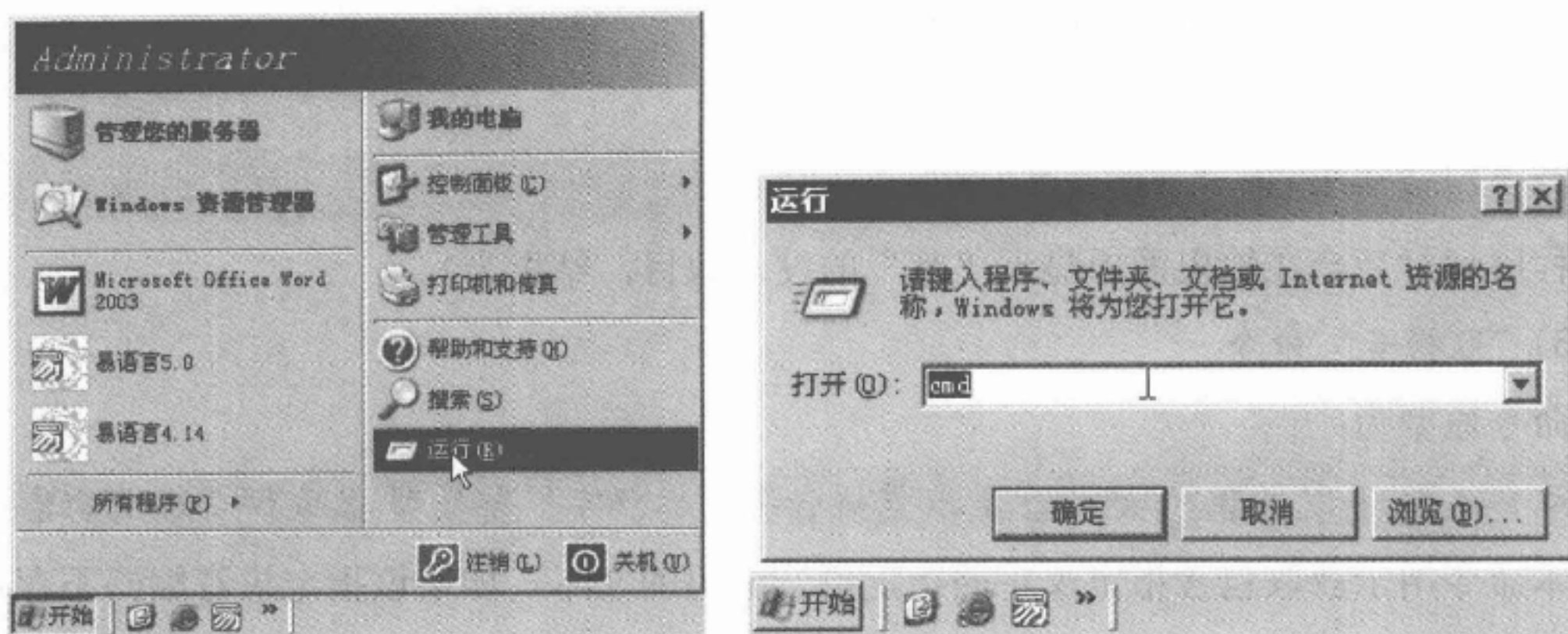


图3-4 查看系统环境变量

### 5) “写环境变量”命令

命令原型为：

<逻辑型> 写环境变量 (文本型 环境变量名称, 文本型 欲写入内容)

本命令用于修改或建立指定的操作系统环境变量。成功返回真，失败返回假。如果欲写入的环境变量不存在，则自动创建该环境变量并写入本命令的第二个参数。需要注意的是本命令所更改的结果只对所在进程有效，不会影响其他进程及Windows系统环境变量。

## 3.8 拼音处理命令

拼音处理命令只是易语言作为中国本地化编程语言特性中的冰山一角。其中提供的命令可以方便地处理汉语拼音相关的功能，使程序开发更迅速、准确率更高。

### 1) “取所有发音”命令

命令原型为：

<文本型数组> 取所有发音 (文本型 欲取其拼音的汉字)

本命令用于获得一个汉字的所有拼音。被操作汉字的读音数与返回的数组成员数一一对应。命令执行时只处理给定参数中第一个字，其他的将被抛弃。如果命令执行失败，将返回一个0成员的空数组。需要注意的是：本命令不区分拼音的声调。

### 2) “取发音数目”命令

命令原型为：

<整数型> 取发音数目 (文本型 欲取其发音数目的汉字)

本命令用于获取一个汉字的发音数量。命令执行时只处理给定参数中第一个字，其他的将被抛弃。如果成功则返回此汉字的发音数量；失败则返回0。





需要注意的是：本命令不区分拼音的声调。

例如：

输出调试文本(取发音数目 (“都”))

程序运行后会在输出面板显示“都”的发音数目，结果为2。

### 3) “取拼音” 命令

命令原型为：

<文本型> 取拼音 (文本型 欲取其拼音编码的汉字，整数型 欲取拼音编码的索引)

本命令用于获取包含指定汉字的指定拼音编码的文本。如果该指定拼音编码不存在，将返回空文本。多音字的第一个发音为常用音。汉字的发音数可以使用“取发音数目”命令获得。

例如：

输出调试文本(取拼音 (“都”, 1))

程序运行后会在输出面板中显示“都”的第1个读音，结果为“dou”。

### 4) “取声母” 命令

命令原型为：

<文本型> 取声母 (文本型 欲取其拼音编码的汉字，整数型 欲取拼音编码的索引)

本命令用于获取包含指定汉字指定拼音编码的声母部分文本。如果指定拼音编码不存在或该汉字此发音无声母，将返回空文本。

例如：

输出调试文本(取声母 (“都”, 1))

程序运行后会在输出面板显示“都”的第1个读音的声母，结果为“d”。

### 5) “取韵母” 命令

命令原型为：

<文本型> 取韵母 (文本型 欲取其拼音编码的汉字，整数型 欲取拼音编码的索引)

本命令用于获取包含指定汉字指定拼音编码的韵母部分文本。如果指定拼音编码不存在，将返回空文本。

例如：

输出调试文本(取韵母 (“都”, 1))

程序运行后会在输出面板显示“都”的第1个读音的韵母，结果为“ou”。

### 6) “发音比较” 命令

命令原型为：

<逻辑型> 发音比较 (文本型 待比较文本一，文本型 待比较文本二，逻辑型 是否支持南方音，[逻辑型 是否为模糊比较])





本命令用于比较两段文本的发音即文本的全拼，如果发音相同返回“真”，否则返回“假”。本命令不对声调进行比较。

参数<3>为真则支持南方音。如果支持南方音，比较时将认为所有相近的发音都相同。如：n 和 l、z 和 zh、ie 和 ue 等。

参数<4>为“真”时，待比较文本一的发音在待比较文本二的首部被包容时即认为比较通过。

参数<4>为“假”时，如长度不同则直接返回“假”。

例如：

输出调试文本(发音比较(“中国”，“重锅人民”，假,真))

程序运行后会在输出面板显示比较的结果，结果为“真”。

输出调试文本(发音比较(“中国”，“重锅人民”，假,假))

程序运行后会在输出面板显示比较的结果，结果为“假”。

输出调试文本(发音比较(“中国人民”，“重锅”，假,真))

程序运行后会在输出面板显示比较的结果，结果为“假”。

#### 7) “输入字比较”命令

命令原型为：

<逻辑型> 输入字比较 (文本型 待比较拼音输入字, 文本型 待比较普通文本, 逻辑型 是否支持南方音, [整数型 拼音输入字类别])

本命令用于对一个拼音输入字文本与另外一个普通文本比较，如果两者相符，返回真，否则返回假。本命令的第一个参数支持首拼、全拼、双拼及自动判别。

(1) 首拼及全拼类：如：“jsj”匹配“计算机”，同时“jisuanji”也匹配“计算机”。系统自动判别该输入内容具体是首拼方式还是全拼方式；

(2) 双拼类：如：“jisrji”匹配“计算机”，双拼编码的方案与 Windows 系统下的双拼输入法一致；

(3) 自动判别类：使用此类别需在拼音输入字的首部加一个半角字符用作判别：分号(“;”)代表首拼及全拼类，冒号(“:”)代表双拼类，如既不为分号也不为冒号，则默认为首拼及全拼类。如“jsj”、“; jsj”、“; jisuanji”、“:jisrji”都匹配“计算机”。

#### 8) 书写拼音输入字时的一些规则

(1) 为了匹配目标文本中大小写英文字母，在相应位置处需使用对应的大写英文字母。

例如：“jsjA”匹配“计算机a”、“计算机A”。

(2) 为了匹配目标文本中半角数字，在相应位置处需使用该数字。

例如：“jsj1”匹配“计算机1”。





(3) 为了匹配目标文本中汉字符号或不知道发音的汉字，在相应位置处需使用该汉字。

例如：“j算j1”匹配“计算机1”。

(4) 拼音输入内容不能包含除了“\_”以外的所有其他半角符号。

参数<3>为真则支持南方音。如果支持南方音，比较时将认为所有相近的发音都相同。

例如：n 和 l、z 和 zh、ie 和 ue 等。

参数<4>指定了“拼音输入内容类别”，类型为“整数型(int)”，可以被省略。参数值可以为以下常量之一：1（#首拼及全拼）；2（#双拼）；3（#自动判别）。如果省略本参数，默认为“#自动判别”。

例如：

输出调试文本(输入字比较(“zhongguo”，“中国”，假,#首拼及全拼))

程序运行后会在输出面板显示比较的结果，结果为“真”。

## 3.9 文本操作命令

在学习文本操作类命令之前，先来了解一下计算机处理文字的方式和相关的概念。

### 3.9.1 文字编码和存储方式

在计算机中，所有的数据在存储和运算时都要使用二进制数表示（计算机用高电平和低电平分别表示1和0）。例如：象a、b、c、d这样的52个字母（包括大小写）以及0、1、2等数字还有一些常用的符号（\*、#、@等）在计算机中存储时也要使用二进制数来表示，而具体用哪些二进制数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而相互之间即要互相通信又不造成混乱，那么就必须使用相同的编码规则，于是美国有关的标准化组织就出台了所谓的ASCII编码，统一规定了上述常用符号用哪些二进制数来表示。

**注解：**除了ASCII码，常用的编码还有Unicode码等字符编码。每个编码集中编码值所对应的字符不尽相同。

### 3.9.2 ASCII码

ASCII（American Standard Code for Information Interchange，美国信息互换标准代码）是基于拉丁字母的一套电脑编码系统。它主要用于显示现代英语和其他西欧语言。它是现今最通用的单字节编码系统，并等同于国际标准ISO/IEC 646。





ASCII 码使用指定的 7 位或 8 位二进制数组合来表示 128 或 256 种可能的字符。标准 ASCII 码也叫基础 ASCII 码，使用 7 位二进制数来表示所有的大写和小写字母，数字 0 到 9、标点符号以及在美式英语中使用的特殊控制字符。其中：

0~31 及 127（共 33 个）是控制字符或通信专用字符（其余为可显示字符），如控制符：LF（换行）、CR（回车）、FF（换页）、DEL（删除）、BS（退格）、BEL（振铃）等；通信专用字符：SOH（文头）、EOT（文尾）、ACK（确认）等；ASCII 值为 8、9、10 和 13 分别转换为退格、制表、换行和回车字符。它们并没有特定的图形显示，但会依不同的应用程序，而对文本显示有不同的影响。

32~126（共 95 个）是字符（32 是空格），其中 48~57 为 0 到 9 十个阿拉伯数字；

65~90 为 26 个大写英文字母，97~122 为 26 个小写英文字母，其余为一些标点符号、运算符号等。

在标准 ASCII 中，其最高位（b7）用作奇偶校验位。所谓奇偶校验，是指在代码传送过程中用来检验是否出现错误的一种方法，一般分奇校验和偶校验两种。奇校验规定：正确的代码一个字节中 1 的个数必须是奇数，若非奇数，则在最高位 b7 添 1；偶校验规定：正确的代码一个字节中 1 的个数必须是偶数，若非偶数，则在最高位 b7 添 0。

后 128 个称为扩展 ASCII 码，目前许多基于 x86 的系统都支持使用扩展（或“高”）ASCII。扩展 ASCII 码允许将每个字符的第 8 位用于确定附加的 128 个特殊符号字符、外来语字母和图形符号，表 3-1 标准 ASCII 表。

表 3-1 标准 ASCII 表

二进制值	十进制值	十六进制值	缩写/字符	解释
00000000	0	00	NUL(null)	空字符
00000001	1	01	SOH(start of handling)	标题开始
00000010	2	02	STX (start of text)	正文开始
00000011	3	03	ETX (end of text)	正文结束
00000100	4	04	EOT (end of transm-ission)	传输结束
00000101	5	05	ENQ (enquiry)	请求
00000110	6	06	ACK (acknow-ledge)	收到通知
00000111	7	07	BEL (bell)	响铃
00001000	8	08	BS (backsp-acc)	退格
00001001	9	09	HT (horizon-tal tab)	水平制表符
00001010	10	0A	LF (NL line feed, new line)	换行键
00001011	11	0B	VT (vertical tab)	垂直制表符
00001100	12	0C	FF (NP form feed, new page)	换页键
00001101	13	0D	CR (carriage return)	回车键
00001110	14	0E	SO (shift out)	不用切换
00001111	15	0F	SI (shift in)	启用切换
00010000	16	10	DLE (data link escape)	数据链路转义
00010001	17	11	DC1 (device control 1)	设备控制 1







二进制值	十进制值	十六进制值	缩写/字符	解释
00010010	18	12	DC2 (device control 2)	设备控制2
00010011	19	13	DC3 (device control 3)	设备控制3
00010100	20	14	DC4 (device control 4)	设备控制4
00010101	21	15	NAK (negati-ve acknowl-edge)	拒绝接收
00010110	22	16	SYN (synchr-onous idle)	同步空闲
00010111	23	17	ETB (end of trans. block)	传输块结束
00011000	24	18	CAN (cancel)	取消
00011001	25	19	EM (end of medium)	介质中断
00011010	26	1A	SUB (substit-ute)	替补
00011011	27	1B	ESC (escape)	溢出
00011100	28	1C	FS (file separat-or)	文件分割符
00011101	29	1D	GS (group separat-or)	分组符
00011110	30	1E	RS (record separat-or)	记录分离符
00011111	31	1F	US (unit separat-or)	单元分隔符
00100000	32	20		空格
00100001	33	21	!	
00100010	34	22	"	
00100011	35	23	#	
00100100	36	24	\$	
00100101	37	25	%	
00100110	38	26	&	
00100111	39	27	'	
00101000	40	28	(	
00101001	41	29	)	
00101010	42	2A	*	
00101011	43	2B	+	
00101100	44	2C	,	
00101101	45	2D	-	
00101110	46	2E	.	
00101111	47	2F	/	
00110000	48	30	0	
00110001	49	31	1	
00110010	50	32	2	
00110011	51	33	3	
00110100	52	34	4	
00110101	53	35	5	
00110110	54	36	6	
00110111	55	37	7	
00111000	56	38	8	
00111001	57	39	9	
00111010	58	3A	:	
00111011	59	3B	;	
00111100	60	3C	<	
00111101	61	3D	=	





续表

二进制值	十进制值	十六进制值	缩写/字符	解释
00111110	62	3E	>	
00111111	63	3F	?	
01000000	64	40	@	
01000001	65	41	A	
01000010	66	42	B	
01000011	67	43	C	
01000100	68	44	D	
01000101	69	45	E	
01000110	70	46	F	
01000111	71	47	G	
01001000	72	48	H	
01001001	73	49	I	
01001010	74	4A	J	
01001011	75	4B	K	
01001100	76	4C	L	
01001101	77	4D	M	
01001110	78	4E	N	
01001111	79	4F	O	
01010000	80	50	P	
01010001	81	51	Q	
01010010	82	52	R	
01010011	83	53	S	
01010100	84	54	T	
01010101	85	55	U	
01010110	86	56	V	
01010111	87	57	W	
01011000	88	58	X	
01011001	89	59	Y	
01011010	90	5A	Z	
01011011	91	5B	[	
01011100	92	5C	\	
01011101	93	5D	]	
01011110	94	5E	^	
01011111	95	5F	_	
01100000	96	60	`	
01100001	97	61	a	
01100010	98	62	b	
01100011	99	63	c	
01100100	100	64	d	
01100101	101	65	e	
01100110	102	66	f	
01100111	103	67	g	
01101000	104	68	h	
01101001	105	69	i	





续表

二进制值	十进制值	十六进制值	缩写/字符	解释
01101010	106	6A	j	
01101011	107	6B	k	
01101100	108	6C	l	
01101101	109	6D	m	
01101110	110	6E	n	
01101111	111	6F	o	
01110000	112	70	p	
01110001	113	71	q	
01110010	114	72	r	
01110011	115	73	s	
01110100	116	74	t	
01110101	117	75	u	
01110110	118	76	v	
01110111	119	77	w	
01111000	120	78	x	
01111001	121	79	y	
01111010	122	7A	z	
01111011	123	7B	{	
01111100	124	7C		
01111101	125	7D	}	
01111110	126	7E	~	
01111111	127	7F	DEL	删除

**注解:**

(1) 易语言中使用的是十进制ASCII码值。

(2) 常见ASCII码值的大小规律:

①  $0 \sim 9 < A \sim Z < a \sim z$

② 数字比字母要小。如 “7” < “F” ；

③ 数字0比数字9要小，并按0到9顺序递增。如 “3” < “8” ；

④ 字母A比字母Z要小，并按A到Z顺序递增。如 “A” < “Z” ；

⑤ 同个字母的大写字母比小写字母要小32。如 “A” < “a” 。

需要记住几个常见字母的ASCII码大小: “A” 为65; “a” 为97; “0” 为 48。

### 3.9.3 区别键代码和文字编码

在学习中，经常会把键代码和文字编码混淆，这是可以理解的，毕竟都是与文字息息相关的两种常见代码。但我们在使用时要时刻谨记它们是完全不同的。

在了解最常用的文字编码ASCII码后，继续了解一下键代码（也叫键盘扫描码）。

键盘上的每一个键都有两个唯一的数值进行标志。但为什么要用两个数值而不是一个数值呢？这是因为一个键可以被按下，也可以被释放。当一个键按下时，它们产生一个唯





一的数值，当一个键被释放时，同样也会产生一个唯一的数值，我们把这些数值都保存在一张表里面，只要通过查表就可以知道是哪一个键被单击，并且可以知道是被按下还是被释放了。而这些数值在系统中被称为键代码（键盘扫描码）。

通过以上可以看出，键代码是与键盘硬件信号息息相关的；而文字编码是与文字存储相关的。它们所在的领域和起到作用截然不同，如果将它们混淆，经常会得到和预想不同的结果。

**注解：**易语言核心库中的“键0、键1……、A键、B键……”常量和窗口组件中的键盘事件参数都是键代码。而本章所介绍的命令涉及的都是ASCII码。

### 3.9.4 文本操作命令

#### 1) “取文本长度”命令

命令原型为：

<整数型> 取文本长度 (文本型 文本数据)

本命令用于获取一个字符串所占内存的字节数。一个英文半角字符占1个字节；一个汉字占2个字节。

例如：

输出调试文本 (取文本长度 (“中国”))

程序执行后会在输出面板中显示结果，结果为“4”。

#### 2) “取文本左边”命令

命令原型为：

<文本型> 取文本左边 (文本型 欲取其部分的文本, 整数型 欲取出字符的数目)

本命令用于获取指定文本中从左边分割的一部分。

参数<2>指定了欲取得的字节长度。

例如：

输出调试文本 (取文本左边 (“中国”, 2))

程序执行后会在输出面板中显示结果，结果为“中”。

#### 3) “取文本右边”命令

命令原型为：

<文本型> 取文本右边 (文本型 欲取其部分的文本, 整数型 欲取出字符的数目)

本命令用于获取指定文本中从右边分割的一部分。

参数<2>指定了欲取得的字节长度。

例如：

输出调试文本 (取文本右边 (“中国”, 2))

程序执行后会在输出面板中显示结果，结果为“国”。





#### 4) “取文本中间”命令

命令原型为:

<文本型> 取文本中间 (文本型 欲取其部分的文本, 整数型 起始取出位置, 整数型 欲取出字符的数目)

本命令用于获取指定文本中的一部分。

参数<2>指定了欲获取的起始位置 (包括该位置字符)。

参数<3>指定了欲取得的字节长度 (包括起始位置)。

例如:

输出调试文本 (取文本中间 (“我爱你, 中国”, 9, 4))

程序执行后会在输出面板中显示结果, 结果为“中国”。

#### 5) “字符”命令

命令原型为:

<文本型> 字符 (字节型 欲取其字符的字符代码)

本命令用于获取ASCII码所对应的文字。

例如:

输出调试文本 (字符 (65))

程序执行后会在输出面板中显示结果, 结果为“A”。

#### 6) “取代码”命令

命令原型为:

<整数型> 取代码 (文本型 欲取字符代码的文本, [整数型 欲取其代码的字符位置])

本命令用于获取一个文本所对应的ASCII码。

参数<2>指定了欲取ASCII码的字符所在文本中的位置。位置从1开始。

例如:

输出调试文本 (取代码 (“B”, 1))

程序执行后会在输出面板中显示结果, 结果为“66”。

#### 7) “寻找文本”命令

命令原型为:

<整数型> 寻找文本 (文本型 被搜寻的文本, 文本型 欲寻找的文本, [整数型 起始搜寻位置], 逻辑型 是否不区分大小写)

本命令用于确定一段文本在另一段文本中最先出现的位置。寻找方向为从前到后。

参数<3>指定了寻找起始位置 (包括指定的位置)。位置从1开始。如果在另一段文本中找到它, 则返回出现的位置, 否则返回“-1”。

例如:

输出调试文本 (寻找文本 (“zhongguo中国中文编程”, “中国”, 1, 假))





程序执行后会在输出面板中显示结果，结果为“9”。

#### 8) “倒找文本”命令

命令原型为：

<整数型> 倒找文本 (文本型 被搜寻的文本, 文本型 欲寻找的文本, [整数型 起始搜寻位置], 逻辑型 是否不区分大小写)

本命令用于确定一段文本在另一段文本中最先出现的位置。寻找方向为从后到前。

参数<3>指定了寻找起始位置（包括指定的位置）。位置从1开始，如果省略则从文本的最后开始寻找。如果在另一段文本中找到它，则返回出现的位置，否则返回“-1”。

例如：

输出调试文本(倒找文本(“zhongguo中国中文编程”，“中国”，,假))

程序执行后会在输出面板中显示结果，结果为“9”。

#### 9) “到大写”命令

命令原型为：

<文本型> 到大写 (文本型 欲变换的文本)

本命令用于将一段文本内的小写字母转换为大写。

例如：

输出调试文本(到大写(“zhongguo”))

程序执行后会在输出面板中显示结果，结果为“ZHONGGUO”。

#### 10) “到小写”命令

命令原型为：

<文本型> 到小写 (文本型 欲变换的文本)

本命令用于将一段文本内的大写字母转换为小写。

例如：

输出调试文本(到小写(“ZHONGGUO”))

程序执行后会在输出面板中显示结果，结果为“zhongguo”。

#### 11) “到全角”命令

命令原型为：

<文本型> 到全角 (文本型 欲变换的文本)

本命令用于将一段文本中的半角字母、空格或数字变换为全角。

例如：

输出调试文本(到全角(“1',2.30”))

程序执行后会在输出面板中显示结果，结果为“1'，2. 3 ( ) ”。







## 12) “到半角”命令

命令原型为:

&lt;文本型&gt; 到半角 (文本型 欲变换的文本)

本命令用于将一段文本中的全角字母、空格或数字变换为半角。

例如:

输出调试文本(到半角 (“1’, 2. 3 ( ) ”))

程序执行后会在输出面板中显示结果, 结果为 “1’,2.3 ( ) ”。

## 13) “到文本”命令

命令原型为:

&lt;文本型&gt; 到文本 (通用型数组/非数组 待转换的数据)

本命令用于将其他类型的数据转换为文本型数据。

例如:

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 到文本 (123456)

» 输出调试文本 (变量1)

程序执行后将 “123456” 转换为文本型数据并保存到变量1中; 在输出面板中显示变量1的值, 结果为 “123456”。

## 14) “删首空”命令

命令原型为:

&lt;文本型&gt; 删首空 (文本型 欲删除空格的文本)

本命令用于删除一段文本中文字前的空格。

例如:

输出调试文本(删首空 (“      中国 ”))

程序执行后将 “      中国 ” 文本中前面的空格删除并在输出面板中显示结果, 结果为 “中国”。

## 15) “删尾空”命令

命令原型为:

&lt;文本型&gt; 删尾空 (文本型 欲删除空格的文本)

本命令用于删除一段文本中文字最后的空格。

例如:

输出调试文本(删尾空 (“中国      ”))

程序执行后将 “中国      ” 文本中后面的空格删除并在输出面板中显示结果, 结果为 “中国”。





## 16) “删首尾空”命令

命令原型为:

<文本型> 删首尾空 (文本型 欲删除空格的文本)

本命令用于删除一段文本中文字前后的空格。

例如:

输出调试文本(删首尾空 (“      中国      ”))

程序执行后将 “      中国      ” 文本中文字前后的空格删除并在输出面板中显示结果, 结果为 “中国”。

## 17) “删全部空”命令

命令原型为:

<文本型> 删全部空 (文本型 欲删除空格的文本)

本命令用于删除一段文本中所有的空格。

例如:

输出调试文本(删全部空 (“      中      国      ”))

程序执行后将 “      中      国      ” 文本中的所有空格删除并在输出面板中显示结果, 结果为 “中国”。

## 18) “文本替换”命令

命令原型为:

<文本型> 文本替换 (文本型 欲被替换的文本, 整数型 起始替换位置, 整数型 替换长度, [文本型 用作替换的文本])

本命令用于将一段文本中指定位置起, 替换指定长度。

参数<4>用于指定用做替换的新文本。

例如:

输出调试文本(文本替换 (“我爱你中国”, 5, 6, “易语言”))

程序执行后将 “我爱你中国” 文本中的第5个字符开始替换6个字符长度, 用作替换的文本为 “易语言”。在输出面板中显示替换后的结果, 结果为 “我爱易语言”。

## 19) “子文本替换”命令

命令原型为:

<文本型> 子文本替换 (文本型 欲被替换的文本, 文本型 欲被替换的子文本, [文本型 用作替换的子文本], [整数型 进行替换的起始位置], [整数型 替换进行的次数], 逻辑型 是否区分大小写)

本命令用作对一段文本中的一串文字组合进行替换。

参数<2>指定了文字的组合。





例如：

输出调试文本(子文本替换 (“AaAaAaAaAaA”, “a”, “B”, 1,, 真))

程序执行后将“AaAaAaAaAaA”中所有的“a”替换为“B”，并将处理后的结果显示在输出面板中。结果为“ABABABABABA”。

例如：

输出调试文本(子文本替换 (“AaAaAaAaAaA”, “a”, “B”, 1,, 假))

结果为“BBBBBBBBBBBBB”。

## 20) “取空白文本” 命令

命令原型为：

<文本型> 取空白文本 (整数型 重复次数)

本命令用于获取指定长度的空白文本。本命令一般用于申请一段空白的内存空间，以便接收来自外部（一般为API调用）的文本。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取空白文本 (10)

» 输出调试文本 (取文本长度 (变量1))

程序执行后“变量1”的长度被改变，并将改变后的长度显示在输出面板中。结果为“10”。

## 21) “取重复文本” 命令

命令原型为：

<文本型> 取重复文本 (整数型 重复次数, 文本型 待重复文本)

本命令用于获取由N段相同文本连接而成的新文本。

例如：

输出调试文本(取重复文本 (3, “易语言”))

程序运行后，“易语言”被复制了3份，并连接为一个文本显示到输出面板中。显示的内容为“易语言易语言易语言”。

## 22) “文本比较” 命令

命令原型为：

<整数型> 文本比较 (文本型 待比较文本一, 文本型 待比较文本二, 逻辑型 是否区分大小写)

本命令用于比较两个文本的大小。





规则：首先比较两个文本中相同位置文字的ASCII码值，若不相等则返回比较结果。相等则继续依次比较。到某个文本的最后时如果一个文本长于另一个文本，长的文本就大于短的文本。若长度相同则这两个文本相等，即返回0。

参数<3>如为“假”，则相同的大小写字母比较时认定它们相等。

例如：

输出调试文本 (文本比较 (“a1”，“A1”，假))

程序执行后对“a1”和“A1”进行不区分大小写的比较。比较结果显示在输出面板中。结果为“0”，即它们相等。

23) “分割文本” 命令

命令原型为：

<文本型数组> 分割文本 (文本型 待分割文本, [文本型 用作分割的文本], [整数型 要返回的子文本数目])

本命令用于使用指定文本分割另一个文本。

参数<3>指定了所需要的分割数量。运行中如果实际结果大于此参数，其余的结果将被抛弃，不包含在返回值中。如果不给定参数，默认返回所有结果。

例如：

变量名	类型	静态	数组	备注
文本变量	文本型		0	
计数变量	整数型			

文本变量 = 分割文本 (“1,2,3”, “,”, )

--- 计次循环首 (取数组成员数 (文本变量), 计数变量)

» 输出调试文本 (文本变量 [计数变量])

--- 计次循环尾 ()

程序执行后使用“,”将“1, 2, 3”分割，并将结果存储到“文本变量”中。程序使用“计次循环首”将“文本变量”的每个成员的值显示到输出面板中。

输出结果为：“1”  
                  “2”  
                  “3”

24) “指针到文本” 命令

命令原型为：

<文本型> 指针到文本 (整数型 内存文本指针)

本命令用于取得一个文本指针所指向的文本数据。本命令一般配合外部DLL时使用。



例如：

变量名	类 型	静态	数组	备 注
内存地址	整数型			

内存地址 = VirtualAlloc (0, 1024 × 1024 × 10, #MEM\_COMMIT, #PAGE\_READWRITE)

写到内存 (“易语言”, 内存地址, -1)

→ 输出调试文本 (指针到文本 (内存地址))

VirtualFree (内存地址, 10 × 1024 × 1024, #MEM\_DECOMMIT)

程序运行后会调用Windows API申请10M内存；将“易语言”写入内存；将内存中的数据以文本型读出；释放内存。

本例程源码见随书光盘中 “\图书例程\第三章\其他\写到内存.e”

## 3.10 时间操作命令

时间操作命令有以下几项。

1) “到时间” 命令

命令原型为：

<日期时间型> 到时间 (通用型 欲转换的文本)

本命令用于将其他数据类型转换到日期时间型。所支持的格式如下：

2009年11月19日12时30分25秒 20091119123025

2009/11/19 12:30:25 2009/11/19/12/30/25 2009/11/19/12:30:25

2009-11-19-12-30-25 2009-11-19-12:30:25 2009.11.19 12:30:25

例如：

变量名	类 型	静态	数组	备 注
变量1	日期时间型			

变量1 = 到时间 (“2099.1.1”)

→ 输出调试文本 (变量1)

程序运行后会将“2099.1.1”这个文本型数据转换为日期时间型数据并存储在“变量1”中。随后程序在输出面板中显示变量1的内容。显示结果为：“2099年1月1日”。

2) “增减时间” 命令

命令原型为：

<日期时间型> 增减时间 (日期时间型 时间, 整数型 被增加部分, 整数型 增加值)

本命令用于定量增减一个日期时间型数据或变量的某一单位。本命令在处理增减时会自动进行进位及退位处理。

参数<2>指定了欲增减的单位。欲增减的单位可以是以下常量之一：1 (#年份)；2 (#季度)；3 (#月份)；4 (#周)；5 (#日)；6 (#小时)；7 (#分钟)；8 (#秒)。





参数<3>指定了需增减的值。正数为增加，负数为减少。

例如：

变量名	类 型	静态	数组	备 注
变量1	日期时间型			

变量1 = [2099年1月1日]

变量1 = 增减时间 (变量1, #月份, 15)

» 输出调试文本 (变量1)

程序运行后首先将“变量1”的值改变为2099年1月1日。再将“变量1”中的时间值增加15个月。最后在输出面板中显示改变后的时间。显示结果为“2100年4月1日”。

### 3) “取时间间隔”命令

命令原型为：

<双精度小数型> 取时间间隔 (日期时间型 时间1, 日期时间型 时间2, 整数型 取间隔部分)

本命令用于获取“时间1”与“时间2”的指定单位间的差。若“时间1”大于“时间2”则返回正数，“时间1”小于“时间2”则返回负数。

**注解：**每个星期以星期天为第一天。

参数<3>指定了欲比较两个时间间隔的单位。单位可以是以下常量之一：1（#年份）；2（#季度）；3（#月份）；4（#周）；5（#日）；6（#小时）；7（#分钟）；8（#秒）。

例如：

变量名	类 型	静态	数组	备 注
变量1	双精度小数型			

变量1 = 取时间间隔 ([2099年1月11日3时], [2099年1月5日4时], #周)

» 输出调试文本 (变量1)

程序执行后对比“2099年1月11日3时”与“2099年1月5日4时”之间间隔的周数。并将结果存储在“变量1”中。在输出面板显示“变量1”的值，显示结果为“1”。

### 4) “取某月天数”命令

命令原型为：

<整数型> 取某月天数 (整数型 年份, 整数型 月份)

本命令用于获取某年的某月的天数。本命令会自动处理闰年2月天数。本命令所支持的年份范围为100~9999。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取某月天数 (8000, 2)

» 输出调试文本 (变量1)





程序运行后会计算8000年2月的天数，并将结果存储在“变量1”中。在输出面板显示“变量1”的值，显示结果为“29”。

#### 5) “时间到文本”命令

命令原型为：

<文本型> 时间到文本 (日期时间型 欲转换到文本的时间, [整数型 转换部分])

本命令用于将“日期时间型”数据或变量值转换为“文本型”数据。

参数<2>可指定欲转换的部分，如果不指定默认为全部转换。转换部分可以是以下常量之一：1（#全部转换）；2（#日期部分）；3（#时间部分）。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 时间到文本 ([2099年5月1日22时50分33秒], #时间部分)

» 输出调试文本 (变量1)

程序运行后将“2099年5月1日22时50分33秒”中的时间部分取出转换为文本型数据，并保存在“变量1”中。在输出面板显示“变量1”的值，显示结果为“22时50分33秒”。

#### 6) “取时间部分”命令

命令原型为：

<整数型> 取时间部分 (日期时间型 欲取其部分的时间, 整数型 欲取的时间部分)

本命令用于获取日期时间型数据中某一部分的值。在取星期几时周日为1，周一为2…依次类推。

参数<2>指定了欲获取的部分。该参数可以是以下常量之一：1（#年份）；2（#季度）；3（#月份）；4（#自年首周数）；5（#日）；6（#小时）；7（#分钟）；8（#秒）；9（#星期几）；10（#自年首天数）。其中：自年首周数、自年首天数均从1开始。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取时间部分 ([2010年9月8日], #星期几)

» 输出调试文本 (变量1)

程序执行后在内部计算“2010年9月8日”是星期几，并将结果保存到“变量1”中。在输出面板显示“变量1”的值，结果为“4”。

#### 7) “取年份”命令

命令原型为：

<整数型> 取年份 (日期时间型 时间)

本命令用于获取一个日期时间型数据的年份部分。





例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取年份 ([2010年9月8日])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日”的年份部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“2010”。

#### 8) “取月份”命令

命令原型为：

<整数型> 取月份 (日期时间型 时间)

本命令用于获取一个日期时间型数据的月份部分。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取月份 ([2010年9月8日])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日”的月份部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“9”。

#### 9) “取日”命令

命令原型为：

<整数型> 取日 (日期时间型 时间)

本命令用于获取一个日期时间型数据的日（几号）部分。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取日 ([2010年9月8日])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日”的日（几号）部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“8”。

#### 10) “取星期几”命令

命令原型为：

<整数型> 取星期几 (日期时间型 时间)

本命令用于获取一个日期时间型数据的星期部分。

**注解：**根据外国的习惯一周是从周日开始的，所以周日的值为1，周一的值为2……依次类推。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取星期几 ([2010年9月8日])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日”的星期部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“4”，即星期三。

### 11) “取小时”命令

命令原型为：

<整数型> 取小时 (日期时间型 时间)

本命令用于获取一个日期时间型数据的小时部分（24时格式）。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取小时 ([2010年9月8日2时30分58秒])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日2时30分58秒”的小时部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“2”。

### 12) “取分钟”命令

命令原型为：

<整数型> 取分钟 (日期时间型 时间)

本命令用于获取一个日期时间型数据的分钟部分。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取分钟 ([2010年9月8日2时30分58秒])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日2时30分58秒”的分钟部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“30”。

### 13) “取秒”命令

命令原型为：

<整数型> 取秒 (日期时间型 时间)

本命令用于获取一个日期时间型数据的秒钟部分。

例如：



变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取秒 ([2010年9月8日2时30分58秒])

» 输出调试文本 (变量1)

程序执行后会获取“2010年9月8日2时30分58秒”的秒钟部分存储到“变量1”中。并在输出面板显示“变量1”的值，显示结果为“58”。

#### 14) “指定时间” 命令

命令原型为：

<日期时间型> 指定时间 (整数型 年, [整数型 月], [整数型 日], [整数型 小时], [整数型 分钟], [整数型 秒])

本命令用于获取一个特定时间的日期时间型数据。

例如：

变量名	类 型	静态	数组	备 注
变量1	日期时间型			

变量1 = 指定时间 (2010, 3, 5, 8, 9, 30)

» 输出调试文本 (变量1)

程序执行后会生成一个“2010年3月5日8时9分30秒”的日期时间型数据，并保存到“变量1”中。在输出面板中显示“变量1”的值，显示结果为“2010年3月5日8时9分30秒”。

#### 15) “取现行时间” 命令

命令原型为：

<日期时间型> 取现行时间 ()

本命令用于获取当前系统的日期和时间。

例如：

变量名	类 型	静态	数组	备 注
变量1	日期时间型			

变量1 = 取现行时间 ()

» 输出调试文本 (变量1)

程序运行后会将当前系统时间取出并存放在“变量1”中。并在输出面板中显示“变量1”的值。结果为执行时的系统时间。

#### 16) “置现行时间” 命令

命令原型为：

<逻辑型> 置现行时间 (日期时间型 欲设置的时间)



本命令用于改变系统当前的时间。

例如：

置现行时间 ([2010年9月8日])

» 输出调试文本 (取现行时间 ())

程序执行后，系统时间被设置为“2010年9月8日”。并在输入窗口显示当前系统时间，显示结果为“2010年9月8日”。

17) “取日期”命令

命令原型为：

<日期时间型> 取日期 (日期时间型 时间)

本命令用于获取日期时间型数据中的日期部分，其小时、分钟、秒被固定设置为0时0分0秒。

例如：

变量名	类型	静态	数组	备注
变量1	日期时间型			

变量1 = 取日期 ([2000年5月4日5时4分6秒])

» 输出调试文本 (变量1)

程序执行后会将“2000年5月4日5时4分6秒”的日期部分取出并保存到“变量1”中。在输出面板中显示“变量1”的值，显示结果为“2000年5月4日”。

18) “取时间”命令

命令原型为：

<日期时间型> 取时间 (日期时间型 时间)

本命令用于获取日期时间型数据中的时间部分，其年、月、日被固定设置为2000年1月1日。

例如：

变量名	类型	静态	数组	备注
变量1	日期时间型			

变量1 = 取时间 ([2000年5月4日5时4分6秒])

» 输出调试文本 (变量1)

程序执行后会将“2000年5月4日5时4分6秒”的时间部分取出并将年份固定在“2000年1月1日”，将组合后的结果保存到“变量1”中。并在输出面板中显示“变量1”的值，显示结果为“2000年1月1日5时4分6秒”。



## 3.11 数值转换命令

数值转换命令有以下几条。

### 1) “到数值”命令

命令原型为：

<双精度小数型> 到数值 (通用型 待转换的文本或数值)

本命令用于将其他类型数据转换为“双精度小数型”数据。

例如：

变量名	类型	静态	数组	备注
变量1	双精度小数型			

变量1 = 到数值 (“123.55”)

→ 输出调试文本 (变量1)

程序运行后文本型“123.55”将被转换为双精度小数型数据并保存到“变量1”中。并在输出面板中显示“变量1”的值,结果为“123.55”。

### 2) “数值到大写”命令

命令原型为：

<文本型> 数值到大写 (双精度小数型 欲转换形式的数值, 逻辑型 是否转换为简体)

本命令用于将双精度小数型数据转换为中文大写文本。

参数<2>指定了返回时文本的样式。若为“真”，则返回表示数值的汉字类型。若为“假”，则返回表示金额的汉字类型。

**注解：**小数部分只支持两位小数。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 数值到大写 (123.55, 假)

→ 输出调试文本 (变量1)

程序运行后会将“123.55”转换为文本型汉字表示方式并保存在“变量1”中。在输出面板显示“变量1”的值,显示结果为“壹佰贰拾叁点伍伍”。

### 3) “数值到金额”命令

命令原型为：

<文本型> 数值到金额 (双精度小数型 欲转换形式的数值, 逻辑型 是否转换为简体)

本命令用于将双精度小数型数据转换为以人民币作单位的汉字书写方式。本命令在处理时只处理到百分位,千分位将被四舍五入。





例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 数值到金额 (123.5665, 假)

» 输出调试文本 (变量1)

程序运行后会将“123.5665”转换为人民币汉字书写方式的文本，并保存在“变量1”中。在输出面板中显示“变量1”的内容，显示结果为：“壹佰贰拾叁元伍角柒分”。

#### 4) “数值到格式文本”命令

命令原型为：

<文本型> 数值到格式文本 (双精度小数型 欲转换为文本的数值, [整数型 小数保留位数], 逻辑型 是否进行千分位分隔)

本命令用于将一个双精度小数型数据转换为指定格式的文本。

参数<2>指定了转换时保留小数的位数。如果大于0，表示小数点右边应四舍五入保留的位数；如果等于0，表示舍入到整数；如果小于0，表示小数点左边舍入到的位置。如：数值到格式文本 (1056.65, 1, 假) 返回 “1056.7”；数值到格式文本 (1056.65, 0, 假) 返回 “1057”；数值到格式文本 (1056.65, -1, 假) 返回 “1060”。如果省略本参数，则默认为保留所有实际存在的小数位。

参数<3>指定了是否进行每三位整数（从右向左数）使用逗号进行分隔。若为“真”则使用，否则不使用。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 数值到格式文本 (6123.5665, , 真)

» 输出调试文本 (变量1)

程序运行后会将“6123.5665”转换为文本型数据，并为每3位整数添加一个逗号做分隔。将转换的结果保存在“变量1”中。在输出面板显示“变量1”的内容，显示结果为“6,123.5665”。

#### 5) “取十六进制文本”命令

命令原型为：

<文本型> 取十六进制文本 (整数型 欲取进制文本的数值)

本命令用于将一个整数型数据转换为十六进制数的文本。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取十六进制文本 (123)

» 输出调试文本 (变量1)





程序运行后会将“123”转换为十六进制数文本，并保存在“变量1”中。在输出面板显示“变量1”的值，显示结果为：“7B”。

6) “取八进制文本”命令

命令原型为：

<文本型> 取八进制文本 （整数型 欲取进制文本的数值）

本命令用于将一个整数型数据转换为八进制数的文本。

例如：

变量名	类 型	静态	数组	备 注
变量1	文本型			

变量1 = 取八进制文本 (123)

» 输出调试文本 (变量1)

程序运行后会将“123”转换为八进制数的文本，并保存在“变量1”中。在输出面板显示“变量1”的值，显示结果为：“173”。

7) “到字节”命令

命令原型为：

<字节型> 到字节 （通用型 待转换的文本或数值）

本命令用于将其他数据类型的数据转换为字节型数据。转换中如果有小数，小数部分将被抛弃。本命令支持转换全角文字类型的数字。

例如：

变量名	类 型	静态	数组	备 注
变量1	字节型			

变量1 = 到字节 (“123”)

» 输出调试文本 (变量1)

程序运行后会将文本型“123”换转为字节型“123”并保存在“变量1”中。在输出面板显示“变量1”的值，显示结果为“123”。

8) “到短整数”命令

命令原型为：

<短整数型> 到短整数 （通用型 待转换的文本或数值）

本命令用于将其他数据类型的数据转换为短整数型数据。转换中如果有小数，小数部分将被抛弃。本命令支持转换全角文字类型的数字。

例如：

变量名	类 型	静态	数组	备 注
变量1	短整数型			

变量1 = 到短整数 (“123.33”)

» 输出调试文本 (变量1)





程序运行后会将文本型“123.33”转换为短整数型“123”，小数部分将被抛弃。将转换后的结果保存在“变量1”中。在输出面板中显示“变量1”的内容，显示结果为“123”。

#### 9) “到整数”命令

命令原型为：

<整数型> 到整数 (通用型 待转换的文本或数值)

本命令将用于将其他数据类型的数据转换为整数型数据。转换中如果有小数，小数部分将被抛弃。本命令支持转换全角文字类型的数字。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 到整数 (“123.33”)

→ 输出调试文本 (变量1)

程序运行后会将文本型“123.33”转换为整数型“123”，小数部分将被抛弃。将转换后的结果保存在“变量1”中。在输出面板显示“变量1”的内容，显示结果为“123”。

#### 10) “到长整数”命令

命令原型为：

<长整数型> 到长整数 (通用型 待转换的文本或数值)

本命令用于将其他数据类型的数据转换为长整数型数据。转换中如果有小数，小数部分将被抛弃。本命令支持转换全角文字类型的数字。

例如：

变量名	类型	静态	数组	备注
变量1	长整数型			

变量1 = 到长整数 (“123.33”)

→ 输出调试文本 (变量1)

程序运行后会将文本型“123.33”转换为长整数型“123”，小数部分将被抛弃。将转换后的结果保存在“变量1”中。在输出面板显示“变量1”的内容，显示结果为“123”。

#### 11) “到小数”命令

命令原型为：

<小数型> 到小数 (通用型 待转换的文本或数值)

本命令用于将其他数据类型的数据转换为小数型数据。本命令支持转换全角文字类型的数字。





例如：

变量名	类 型	静态	数组	备 注
变量1	小数型			

变量1 = 到小数 (“5 6 7 . 8”)  
 输出调试文本 (变量1)

程序运行后会将“5 6 7 . 8”转换为小数型数据，并保存在“变量1”中。在输出面板中显示“变量1”的值，显示结果为“567.8”。

### 3.12 字节集操作命令

字节集，即字节的集合。字节集相当于一个字节型的数组。所以字节集可以和字节型数组相互替换。

1) “取字节集长度”命令

命令原型为：

<整数型>	取字节集长度 (字节集 字节集数据)
-------	--------------------

本命令用于获得一个字节集数据的长度，也可以理解为获取一个字节型数组的成员数。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取字节集长度 ({ 3, 5, 6 })  
 输出调试文本 (变量1)

程序运行后会获取“{ 3, 5, 6 }”的长度并保存在“变量1”中。在输出面板中显示“变量1”的值，显示结果为“3”。

2) “到字节集”命令

命令原型为：

<字节集>	到字节集 (通用型数组/非数组 欲转换为字节集的数据)
-------	-----------------------------

本命令用于将其他数据类型的数据转换为字节集型。

例如：

变量名	类 型	静态	数组	备 注
变量1	字节集			

变量1 = 到字节集 (“易语言”)  
 输出调试文本 (取字节集长度 (变量1))





程序运行后会将“易语言”转换为字节集型数据，并保存在“变量1”中。在输出面板中显示“变量1”的长度；因为每个汉字占2个字节，所以显示结果为“6”。

注：这时“变量1”的内容为{210, 215, 211, 239, 209, 212}。

### 3) “取字节集数据”命令

命令原型为：

<通用型> 取字节集数据 (字节集 欲取出其中数据的字节集，整数型 欲取出数据的类型，[整数型 起始索引位置])

本命令用于将字节集型数据转换为指定数据类型。本命令会自动处理需要取出的数据长度。

参数<2>指定了欲转换到的数据类型。返回值的类型也会随之改变。

参数<3>指定了起始处理的位置，位置从1开始。如果使用一个变量作为参数，则命令执行后会改变这个变量的值为读取后停留在的数据位置。如果移动后到达字节集的末尾，将修改该变量的内容为-1。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取字节集数据 ({ 0, 0, 1, 23 }, #整数型, )

» 输出调试文本 (变量1)

程序运行后会将字节集数据“{ 0, 0, 1, 23 }”转换为整数型，并保存在“变量1”中。在输出面板显示“变量1”的内容，显示结果为“385941504”。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取字节集数据 ({ 210, 215, 211, 239, 209, 212 }, #文本型, )

» 输出调试文本 (变量1)

程序运行后会将字节集数据“{ 210, 215, 211, 239, 209, 212 }”转换为文本型，并保存在“变量1”中。在输出面板显示“变量1”的内容，显示结果为“易语言”。

### 4) “取字节集左边”命令

命令原型为：

<字节集> 取字节集左边 (字节集 欲取其部分的字节集，整数型 欲取出字节的数目)

本命令用于获取一个字节集数据的一部分，截取方式为由左向右。起始获取位置从最左边（第1位）开始。

参数<2>指定了欲获取的长度。





例如：

变量名	类 型	静态	数组	备 注
变量1	字节集			

变量1 = 取字节集左边 ({ 65, 66, 67 }, 2)  
» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序运行后会在“{ 65, 66, 67 }”中取最左边的两字节数据，并保存到“变量1”中。将“变量1”中的数据转换为文本显示在输出面板中，显示结果为“AB”。

5) “取字节集右边”命令

命令原型为：

<字节集> 取字节集右边 (字节集 欲取其部分的字节集，整数型 欲取出字节的数目)  
本命令用于获取一个字节集数据的一部分，截取方式为由右向左。起始获取位置从最右边（最后1位）开始。

参数<2>指定了欲获取的长度。

例如：

变量名	类 型	静态	数组	备 注
变量1	字节集			

变量1 = 取字节集右边 ({ 65, 66, 67 }, 2)  
» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序运行后会在“{ 65, 66, 67 }”中取最右边的两字节数据，并保存到“变量1”中。将“变量1”中的数据转换为文本显示在输出面板中，显示结果为“BC”。

6) “取字节集中间”命令

命令原型为：

<字节集> 取字节集中间 (字节集 欲取其部分的字节集，整数型 起始取出位置，整数型 欲取出字节的数目)  
本命令用于获取一个字节集数据的一部分，截取方式为由左向右。  
参数<2>指定了起始获取位置。  
参数<3>指定了欲获取的长度。

例如：

变量名	类 型	静态	数组	备 注
变量1	字节集			

变量1 = 取字节集中间 ({ 65, 66, 67 }, 2, 1)  
» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序运行后会在“{ 65, 66, 67 }”中从第2个字节起（包括第2个）取出1个字节的数  
据，并保存到“变量1”中。将“变量1”中的数据转换为文本显示在输出面板中，显示结  
果为“B”。





## 7) “寻找字节集”命令

命令原型为:

<整数型> 寻找字节集 (字节集 被搜寻的字节集, 字节集 欲寻找的字节集, [整数型 起始搜寻位置])

本命令用于取得一个字节集在另一个字节集中的位置, 寻找顺序为由左到右。若在另一个字节集中找到它, 则命令返回它所出现的位置, 否则返回-1。

参数<3>指定了起始寻找的位置。需要注意的是, 如果找到被寻找的数据, 该命令将立即返回, 不论被寻找的数据在下面是否再出现。欲寻找所有出现的位置, 请使用循环和改变本命令的第三个参数来实现。

例如:

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 寻找字节集 ({ 11, 22, 12, 13, 22 }, { 12, 13 }, )

» 输出调试文本 (变量1)

程序执行后, 在“{ 11, 22, 12, 13, 22 }”中寻找“{ 12, 13 }”的位置。并将结果保存在“变量1”中。在输出面板显示“变量1”的内容, 显示结果为“3”。

## 8) “倒找字节集”命令

命令原型为:

<整数型> 倒找字节集 (字节集 被搜寻的字节集, 字节集 欲寻找的字节集, [整数型 起始搜寻位置])

本命令用于取得一个字节集在另一个字节集中的位置, 寻找顺序为由右到左。若在另一个字节集中找到它, 则命令返回它所出现的位置, 否则返回-1。

参数<3>指定了起始寻找的位置, 若省略, 默认为从数据的最后开始寻找。需要注意的是, 如果找到被寻找的数据, 该命令将立即返回, 不论被寻找的数据在下面是否再出现。欲寻找所有出现的位置, 请使用循环和改变本命令的第三个参数来实现。

例如:

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 倒找字节集 ({ 11, 22, 12, 13, 22 }, { 12, 13 }, )

» 输出调试文本 (变量1)

程序执行后, 在“{ 11, 22, 12, 13, 22 }”中寻找“{ 12, 13 }”的位置。并将结果保存在“变量1”中。在输出面板显示“变量1”的内容, 显示结果为“3”。

## 9) “字节集替换”命令

命令原型为:





<字节集> 字节集替换 (字节集 欲替换其部分的字节集, 整数型 起始替换位置, 整数型 替换长度, [字节集 用作替换的字节集] )

本命令用于将一个字节集数据中的一部分使用新数据替换。

参数<2>指定了欲操作的起始位置。

参数<3>指定了从起始位置起欲替换的长度。

参数<4>指定了用作替换的新数据。若第四个参数被省略, 则欲替换的部分将被删除。

例如:

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 字节集替换 ({ 65, 66, 67, 68, 69 }, 2, 3, { 56, 56, 56 })

» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序执行后将 “{ 65, 66, 67, 68, 69 }” 从第2个字节开始使用 “{ 56, 56, 56 }” 替换3个字节长的数据。将替换后的结果 (结果为 “{ 65, 56, 56, 56, 69 }”) 保存在 “变量1” 中; 将 “变量1” 中的数据转换为文本显示在输出面板中, 输出结果为 “A888E”。

10) “子字节集替换” 命令

命令原型为:

<字节集> 子字节集替换 (字节集 欲被替换的字节集, 字节集 欲被替换的子字节集, [字节集 用作替换的子字节集], [整数型 进行替换的起始位置], [整数型 替换进行的次数] )

本命令用于将一个字节集中的指定特征数据替换为其他数据。

参数<2>指定了欲被替换数据的特征。

参数<3>指定了替换所使用的新数据。

参数<4>指定了欲替换的起始位置。

参数<5>指定了进行替换的次数; 如果该参数被省略, 则替换起始位置后所有符合条件的数据。

例如:

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 子字节集替换 ({ 65, 66, 66, 66, 69 }, { 66 }, { 56 }, , )

» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序执行后将 “{ 65, 66, 66, 66, 69 }” 中的 “{ 66 }” 全部替换为 “{ 56 }”, 并将替换后的结果 (结果为 “{ 65, 56, 56, 56, 69 }”) 保存在 “变量1” 中; 将 “变量1” 中的数据转换为文本显示在输出面板中, 输出结果为 “A888E”。





## 11) “取空白字节集”命令

命令原型为:

&lt;字节集&gt; 取空白字节集 (整数型 零字节数目)

本命令用于获取指定长度的空白字节集。字节集中每个字节的值为0。本命令一般用于预先申请指定长度的内存空间，以便在调用外部DLL命令时使用。

例如:

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 取空白字节集 (50)

» 输出调试文本 (取字节集长度 (变量1))

程序执行后将生成长度为50的空字节集数据并保存在“变量1”中。在输出面板中显示“变量1”的长度，显示结果为“50”。

## 12) “取重复字节集”命令

命令原型为:

&lt;字节集&gt; 取重复字节集 (整数型 重复次数, 字节集 待重复的字节集)

本命令用于由指定数量的重复数据生成的字节集。

例如:

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 取重复字节集 (10, { 66 })

» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序执行后将生成由10个“{ 66 }”所组成的新数据，并保存在“变量1”中。将“变量1”中的数据转换为文本型显示在输出面板中，显示结果为“BBBBBBBBBBB”。

## 13) “分割字节集”命令

命令原型为:

<字节集数组> 分割字节集 (字节集 待分割字节集, [字节集 用作分割的字节集], [整数型 要返回的子字节集数目])

本命令用于将一个字节集使用指定特征的数据进行分割。但用作分割的数据不会包含在返回值中。

例如:

变量名	类型	静态	数组	备注
变量1	字节集		0	

变量1 = 分割字节集 ({ 1, 0, 2, 3, 0, 4, 0, 5 }, { 0 }, )

» 输出调试文本 (取数组成员数 (变量1))





程序执行后会将“{ 1, 0, 2, 3, 0, 4, 0, 5 }”使用“{ 0 }”进行分割。并将分割后的结果保存在“变量1”中。在输出面板显示“变量1”的数组成员数，即被分割的数量。显示结果为“4”。

#### 14) “指针到字节集”命令

命令原型为：

<字节集> 指针到字节集 (整数型 内存数据指针, 整数型 内存数据长度)

本命令用于取得一个内存指针所指向的字节集数据。

参数<2>指定了欲取得数据的长度。本命令一般配合外部DLL使用。

## 3.13 磁盘操作命令

### 3.13.1 相关知识

#### 1) 绝对路径与相对路径

在计算机中找寻需要的文件就必须知道文件的位置，而表示文件位置的方式就是路径。例如路径：“D:\编程\易语言.exe”，可以知道“易语言.exe”文件是在D盘的“编程”目录中。类似这样完整描述文件位置的路径就是“绝对路径”。

什么是“相对路径”？“相对路径”就是指由这个文件（或程序）所在的路径引起的跟其他文件（或文件夹）的路径关系。使用相对路径可以为我们带来非常多的便利，最大的好处就是在引用本程序相关的外部文件（如图片，声音）时，不再需要知道程序所在的完整路径就可以直接访问所需的外部文件。

#### 2) 当前目录

当前目录即使用相对路径时的系统起始装载位置。当前目录是程序中一个临时的变量，可以使用相关命令来获取和修改它的值。

### 3.13.2 易语言中的磁盘操作命令

#### 1) “取磁盘总空间”命令

命令原型为：

<整数型> 取磁盘总空间 ([文本型 磁盘驱动器字符])

本命令用于获取指定分区磁盘的总容量。返回以 1024 字节 (KB) 为单位的指定磁盘全部空间。如果失败，返回-1。

参数<1>指定了欲获取分区的盘符，若不指定则返回本程序所在磁盘的总空间。







例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取磁盘总空间 0

→ 输出调试文本 (变量1)

程序运行后会获取程序所在磁盘的总容量，并保存在“变量1”中。在输出面板中显示“变量1”的值。

## 2) “取磁盘剩余空间”命令

命令原型为：

<整数型> 取磁盘剩余空间 ([文本型 磁盘驱动器字符])

本命令用于获取指定分区磁盘的剩余空间。返回以 1024 字节 (KB) 为单位的指定磁盘现行剩余空间。如果失败，返回-1。

参数<1>指定了欲获取分区的盘符，若不指定则返回本程序所在磁盘的剩余空间。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取磁盘剩余空间 0

→ 输出调试文本 (变量1)

程序运行后会获取程序所在的磁盘剩余容量，并保存在“变量1”中。在输出面板中显示“变量1”的值。

## 3) “取磁盘卷标”命令

命令原型为：

<文本型> 取磁盘卷标 ([文本型 磁盘驱动器字符])

本命令用于获取指定磁盘的卷标。

参数<1>指定了欲获取卷标的盘符，若不指定则返回本程序所在磁盘的卷标。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取磁盘卷标 0

→ 输出调试文本 (变量1)

程序运行后会取得本程序所在磁盘的卷标，并保存在“变量1”中。在输出面板中显示“变量1”的值。

## 4) “置磁盘卷标”命令

命令原型为：

<逻辑型> 置磁盘卷标 ([文本型 磁盘驱动器字符], 文本型 欲置入的卷标文本)

本命令用于改变一个分区的磁盘卷标。执行成功返回“真”，失败返回“假”。



参数<1>指定了欲改变卷标的分区。若不指定，则改变程序所在分区的卷标。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

置磁盘卷标 ( “易语言” )

变量1 = 取磁盘卷标 ()

→ 输出调试文本 (变量1)

程序执行后将程序所在的分区卷标设置为“易语言”；获取程序所在分区的卷标保存在“变量1”中；在输出面板中显示“变量1”的内容；显示结果为“易语言”。

#### 5) “改变驱动器”命令

命令原型为：

<逻辑型> 改变驱动器 (文本型 欲改变到的驱动器)

本命令用于改变“当前目录”的盘符。若成功改变，则“当前目录”为指定盘符的根路径，命令返回“真”。否则“当前目录”不变，命令返回“假”。“当前目录”的初始值为程序所在路径。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

改变驱动器 (“D”)

变量1 = 取当前目录 ()

→ 输出调试文本 (变量1)

程序运行后将当前目录改变为“D”盘根目录下（假设D盘存在）。获取当前路径并保存在“变量1”中。在输出面板中显示“变量1”的内容，显示结果为“D:\”。

#### 6) “改变目录”命令

命令原型为：

<逻辑型> 改变目录 (文本型 欲改变到的目录)

本命令用于改变“当前目录”的路径。若命令执行成功，“当前目录”被改变，命令返回“真”；否则“当前目录”不变，命令返回“假”。

参数<1>指定了欲改变到的路径。如果这个路径是带盘符的，那么将视为绝对路径，否则将视为相对路径。使用命令时应先确认被改变后的路径是否存在，如果被改变的目标路径不存在，命令返回为“假”。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

改变驱动器 (“D”)

改变目录 (“a”)

变量1 = 取当前目录 ()

→ 输出调试文本 (变量1)





程序运行后将当前目录的盘符改变为“D”盘根目录下（假设D盘存在）。改变当前目录为“a”（假设D:\a存在）文件夹中。获取当前目录的路径并保存在“变量1”中。在输出面板显示“变量1”的内容，结果为“D:\a”。

#### 7) “取当前目录”命令

命令原型为：

<文本型> 取当前目录 ()

本命令用于获取程序使用的当前的默认路径。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取当前目录 ()

» 输出调试文本 (变量1)

程序运行后将取得程序当前的默认路径并保存在“变量1”中。在输出面板显示“变量1”的内容。

#### 8) “创建目录”命令

命令原型为：

<逻辑型> 创建目录 (文本型 欲创建的目录名称)

本命令用于创建一个指定名称的目录。创建成功返回“真”，失败返回“假”。

参数<1>指定了新文件夹的名称。若名称中包含盘符，则被视为绝对路径；否则将被视为当前目录的相对路径。需要注意的是，本命令一次只能创建一级目录，不能一次创建多级目录。

例如：

创建目录 (“C:\易语言”)

程序运行后会在C盘（假设C盘存在）下建立名为“易语言”的目录。

#### 9) “删除目录”命令

命令原型为：

<逻辑型> 删除目录 (文本型 欲删除的目录名称)

本命令用于删除一个指定名称的目录和其中的所有内容。删除成功返回“真”，失败返回“假”。

参数<1>指定了欲删除文件夹的名称。若名称中包含盘符，则被视为绝对路径；否则将被视为当前目录的相对路径。

例如：

删除目录 (“C:\易语言”)

程序运行后会删除C盘中的“易语言”目录。



## 10) “复制文件”命令

命令原型为:

<逻辑型> 复制文件 (文本型 被复制的文件名, 文本型 复制到的文件名)

本命令用于创建一个已存在文件的拷贝。若执行成功, 命令返回“真”, 否则返回“假”。

参数<1>指定了被复制的文件名。如果这个文件名包含盘符则被视为绝对路径; 否则被视为当前目录的相对路径。

参数<2>指定了复制产生出的新文件的名称。如果这个文件名包含盘符, 则被视为绝对路径; 否则被视为当前目录的相对路径。

例如:

改变目录 (“C:\”)

复制文件 (“adc.exe”, “123.exe”)

程序运行后将当前目录改变为“C:\”。复制当前目录下的“adc.exe”, 复制出的新文件名为“123.exe”(这里假设所使用的文件及路径存在)。

## 11) “移动文件”命令

命令原型为:

<逻辑型> 移动文件 (文本型 被移动的文件, 文本型 移动到的位置)

本命令用于改变一个文件所在的路径。若执行成功, 命令返回“真”, 否则返回“假”。

参数<1>指定了被移动的文件名。如果这个文件名包含盘符则被视为绝对路径; 否则被视为当前目录的相对路径。

参数<2>指定了移动后的新文件的名称。如果这个文件名包含盘符, 则被视为绝对路径; 否则被视为当前目录的相对路径。

例如:

改变目录 (“C:\”)

移动文件 (“adc.exe”, “D:\123.exe”)

程序运行后将当前目录改变为“C:\”。将当前目录下的“adc.exe”移动到“D:\”, 移动后的新文件名为“123.exe”(这里假设所使用的文件及路径存在)。

## 12) “删除文件”命令

命令原型为:

<逻辑型> 删除文件 (文本型 欲删除的文件名)

本命令用于删除一个指定文件。若执行成功, 命令返回“真”, 否则返回“假”。

参数<1>指定了欲删除的文件名。如果这个文件名包含盘符则被视为绝对路径; 否则被视为当前目录的相对路径。需要注意的是, 使用本命令删除后不会移动到回收站。

例如:

删除文件 (“D:\123.exe”)





程序执行后会彻底删除“D:\123.exe”（这里假设D:\123.exe存在）。

### 13) “文件更名”命令

命令原型为：

<逻辑型> 文件更名（文本型 欲更名的原文件或目录名，文本型 欲更改为的现文件或目录名）

本命令用于改变一个文件或文件夹的名称。若执行成功，命令返回“真”，否则返回“假”。

参数<1>指定了欲删除的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。本命令也可用作移动一个文件或文件夹。

例如：

改变目录（“C:\”）

文件更名（“adc.exe”，“D:\123.exe”）

程序运行后将当前目录改变为“C:\”。将当前目录下的“adc.exe”移动到“D:\”，移动后的改变其文件名为“123.exe”（这里假设所使用的文件及路径存在）。

### 14) “文件是否存在”命令

命令原型为：

<逻辑型> 文件是否存在（文本型 欲测试的文件名称）

本命令用于检查指定文件是否存在。如果不存在，命令返回“假”，存在返回“真”。

例如：

变量名	类型	静态	数组	备注
变量1	逻辑型			

变量1 = 文件是否存在（“c:\123.exe”）  
» 输出调试文本（变量1）

程序运行后会检测“C:\123.exe”是否存在，并将结果保存在“变量1”中。在输出面板中显示“变量1”的值。

### 15) “寻找文件”命令

命令原型为：

<文本型> 寻找文件（[文本型 欲寻找的文件或目录名称]，[整数型 欲寻找文件的属性]）

本命令用于寻找满足指定条件的文件或文件夹。找到后返回其文件名，找不到返回空文本（“”）。

参数<1>用于指定被寻找的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。此参数支持通配符“\*”和“?”。

参数<2>指定了欲寻找文件的属性。文件属性可以是以下常量之一或几个常量的组合：1（#只读文件）；2（#隐藏文件）；4（#系统文件）；16（#子目录）；32（#存档文





件)。通过这些常量值加起来可以一次设置多个文件属性。如果省略本参数，默认为搜寻除子目录外的所有文件。

在使用时，第一次调用本命令时必须指定第一个参数，否则本命令将返回空文本。如果想继续得到满足上一次调用条件的其他文件时，只要再次调用本命令，且不向第一个参数传递任何数据即可。

例如：

变量名	类 型	静态	数组	备 注
变量1	文本型			

```

变量1 = 寻找文件 ("c:\*.*", #隐藏文件)
--> 判断循环首 (变量1 ≠ "")
    输出调试文本 (变量1)
    变量1 = 寻找文件 (, #隐藏文件)
-- 判断循环尾 ()
    
```

程序执行后会将C盘中所有的隐藏文件和隐藏文件夹的名称在输出面板中显示。

本例程源码见随书光盘中“\图书例程\第三章\磁盘操作\枚举文件.e”。

### 16) “取文件时间” 命令

命令原型为：

<日期时间型> 取文件时间 (文本型 文件名)

本命令用于获取指定文件被创建或最后修改后的日期和时间。如果该文件不存在，将返回100年1月1日。

参数<1>指定了欲取得时间的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。

例如：

变量名	类 型	静态	数组	备 注
变量1	日期时间型			

```

改变目录 ("c:\")
变量1 = 取文件时间 ("123.exe")
-- 输出调试文本 (变量1)
    
```

程序执行后，会获取“c:\123.exe”的文件修改时间并保存在“变量1”中。在输出面板显示“变量1”的内容。

### 17) “取文件尺寸” 命令

命令原型为：

<整数型> 取文件尺寸 (文本型 文件名)

本命令用于获取一个文件的大小。命令执行成功返回文件所占用的字节数，失败返回-1。

参数<1>指定了欲取得大小的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。





例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取文件尺寸 (“c:\123.exe”)

→ 输出调试文本 (变量1)

程序执行后会获取 “c:\123.exe” 的大小并保存在 “变量1” 中。在输出面板显示 “变量1” 的内容（这里假设c:\123.exe存在）。

### 18) “取文件属性” 命令

命令原型为：

<整数型> 取文件属性 (文本型 文件名)

本命令用于获取指定文件的属性。命令执行成功返回一个属性或多个属性组合值。文件属性可能为以下常量或几个常量值的组合：

1 (#只读文件)；2 (#隐藏文件)；4 (#系统文件)；16 (#子目录)；32 (#存档文件)。要判断是否设置了某个属性，在返回值与想要得知的属性值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示包含了这个属性值。

参数<1>指定了欲取得属性的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取文件属性 (“c:\123.exe”)

→ 输出调试文本 (变量1)

程序执行后会获取 “c:\123.exe” 所拥有的文件属性，并保存在 “变量1” 中。在输出面板显示 “变量1” 的值（这里假设c:\123.exe存在）。

### 19) “置文件属性” 命令

命令原型为：

<逻辑型> 置文件属性 (文本型 欲设置其属性的文件名称，整数型 欲设置为的属性值)

本命令用于更改指定文件的文件属性。命令执行成功返回“真”，失败返回“假”。

参数<1>指定了设置其属性的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。

参数<2>指定了新的属性。属性可以是以下常量之一或几个常量的组合：

1 (#只读文件)；2 (#隐藏文件)；4 (#系统文件)；32 (#存档文件)。





例如：

置文件属性 (“c:\abc.txt”, #只读文件)

程序执行后会将“c:\abc.txt”设置为只读文件（这里假设c:\abc.txt存在）。

20) “取临时文件名” 命令

命令原型为：

<文本型> 取临时文件名 ([文本型 目录名])

本命令用于获取一个在指定目录中未被使用的临时文件名。

参数<1>指定了目标目录路径。如果被省略，默认为系统的临时目录。需要注意的是，如果指定的目录名不存在，或无法访问，本命令是可以返回正确的文件名，但在使用这个文件名时要注意它的有效性。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取临时文件名 (“z:\”)

» 输出调试文本 (变量1)

程序执行后会获取一个在“z:\”未被使用的临时文件名，并保存在“变量1”中。在输出面板显示“变量1”的值。

21) “读入文件” 命令

命令原型为：

<字节集> 读入文件 (文本型 文件名)

本命令用于将指定文件读入内存。命令执行成功返回所读取的数据，执行失败返回空字节集“{}”。

参数<1>指定了欲读取的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。但在读取较大的文件时应小心内存溢出，建议处理较大文件时使用文件读写类命令。

例如：

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 读入文件 (“c:\abc.txt”)

» 输出调试文本 (取字节集数据 (变量1, #文本型, ))

程序执行后会将“c:\abc.txt”中的数据保存在“变量1”中。将“变量1”中的内容转换为文本显示在输出面板中（这里假设c:\abc.txt存在）。

22) “写到文件” 命令

命令原型为：

<逻辑型> 写到文件 (文本型 文件名, 字节集 欲写入文件的数据, ...)





本命令用于将一段或几段数据写到一个文件中。命令执行成功返回“真”，失败返回“假”。

参数<1>指定了写出的文件名。如果这个文件名包含盘符则被视为绝对路径；否则被视为当前目录的相对路径。如果文件不存在则创建它后写出；如果文件存在原有内容将被覆盖。

例如：

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 到字节集 (“易语言”)

» 写到文件 (“c:\abc.txt”, 变量1)

程序执行后将“易语言”转换为字节集并保存在“变量1”中。将“变量1”中的数据写到“c:\abc.txt”文件中（这里假设c:\abc.txt存在）。

## 3.14 文件读写命令

文件读写命令有以下几条。

### 1) “打开文件”命令

命令原型为：

<整数型> 打开文件 (文本型 欲打开的文件名称, [整数型 打开方式], [整数型 共享方式])

本命令用于打开一个磁盘文件以便进行操作。成功返回被打开文件的文件号，失败返回 0。打开的文件如果不使用应调用“关闭文件”或“关闭所有文件”命令来对所占用的资源进行释放。

参数<2>指定了对文件的操作方式，如果省略本参数，默认为“#读写”。此参数可以为以下常量之一。

1（#读入：从指定文件读入数据，如果该文件不存在则失败）；2（#写出：写出数据到指定文件，如果该文件不存在则失败）；3（#读写：从文件中读入数据或者写出数据，如果该文件不存在则失败）；4（#重写：写出数据到指定文件。如果该文件不存在则先创建一个新文件，如果已经存在就先清除其中的所有数据）；5（#改写：写出数据到指定文件。如果该文件不存在则创建一个新文件，如果已经存在就直接打开）；6（#改读：从文件中读入数据或者写出数据）。如果该文件不存在则创建一个新文件，如果已经存在就直接打开。

参数<3>指定了限制其他进程操作此文件的方式。如果省略此参数，默认为“#无限制”。共享方式值可以为以下常量之一：

1（#无限制：允许其他进程任意读写此文件）；2（#禁止读：禁止其他进程读此文件）；





3 (#禁止写: 禁止其他进程写此文件); 4 (#禁止读写: 禁止其他进程读写此文件)。

例如:

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开文件 ("c:\abc.txt", #重写, #无限制)  
» 输出调试文本 (变量1)  
关闭文件 (变量1)

程序执行后会打开“c:\abc.txt”，并将文件号保存在“变量1”中。在输出面板显示“变量1”的值。关闭“变量1”中文件号所对应的文件。

2) “打开内存文件”命令

命令原型为:

<整数型> 打开内存文件 ()

本命令用于建立一个内存文件，以计算机内存为存储介质对数据进行文件式的快速输入或输出。成功返回被打开文件的文件号，失败返回 0。操作内存文件时需要注意内存文件的大小。如果过大，可能造成程序缓慢或引起内存溢出。打开的文件如果不使用应调用“关闭文件”或“关闭所有文件”命令来对所占用的资源进行释放。

例如:

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开内存文件 ()  
» 输出调试文本 (变量1)  
关闭文件 (变量1)

程序执行后会在内存中建立一个虚拟文件，并返回文件号保存在“变量1”中。在输出面板显示“变量1”的值。关闭“变量1”中文件号所对应的文件。

3) “关闭文件”命令

命令原型为:

<无返回值> 关闭文件 (整数型 欲关闭的文件号)

本命令用于释放被打开的文件资源。如果文件号对应的是一个内存文件，那么这个内存文件所占用的空间将被释放。

参数<1>指定了欲关闭文件对应的文件号。

4) “关闭所有文件”命令

命令原型为:

<无返回值> 关闭所有文件 ()

本命令用于释放所有被打开的文件资源。





例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开内存文件 0

» 输出调试文本 (变量1)

关闭所有文件 0

程序执行后会在内存中建立一个虚拟文件，并返回文件号保存在“变量1”中。在输出面板显示“变量1”的值。关闭所有已打开的文件。

### 5) “锁住文件”命令

命令原型为：

<逻辑型> 锁住文件 (整数型 欲加锁或解锁的文件号, 整数型 欲加锁或解锁的位置, 整数型 欲加锁或解锁的长度, [整数型 加锁重试时间])

本命令用于在有若干个进程访问同一个文件的环境中，使用本命令拒绝其他进程对被打开文件中的某个区域进行读写访问。成功返回“真”，失败返回“假”。

参数<2>指定了欲加锁的起始位置。

参数<3>指定了从第二个参数指定的位置起需要锁住的长度。

参数<4>指定加锁失败后反复进行重试的毫秒数。如果被省略，默认为 0，即一旦失败立即返回。如果参数值提供为 -1，则反复进行重试，直到加锁成功为止。

如果重试时间不为0，则命令所在线程将等待本命令执行完毕再执行下面的代码。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开文件 (“c:\abc.txt”, #重写, #无限制)

锁住文件 (变量1, 0, 100, 0)

程序执行后会锁住“c:\abc.txt”。如果这时使用其他程序试图改变“c:\abc.txt”中0到100字节的内容时，将无法完成。

### 6) “解锁文件”命令

命令原型为：

<逻辑型> 解锁文件 (整数型 欲加锁或解锁的文件号, 整数型 欲加锁或解锁的位置, 整数型 欲加锁或解锁的长度)

本命令用于解除由“锁住文件”命令对文件所进行的锁定。成功返回“真”，失败返回“假”。

**注解：** 本命令调用时所提供的参数值必须与调用“锁住文件”命令时所提供的参数值完全一致。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 打开文件 (“c:\abc.txt”, #重写, #无限制)  
锁住文件 (变量1, 0, 100, 0)  
信息框 (“解锁”, 0, )  
解锁文件 (变量1, 0, 100)

程序运行后会禁止其他程序对“c:\abc.txt”进行更改，并弹出信息框。点击信息框后解除对“c:\abc.txt”的锁定。

7) “移动读写位置” 命令

命令原型为：

<逻辑型> 移动读写位置 (整数型 欲进行操作的文件号, [整数型 起始移动位置], 整数型 移动距离)

本命令用于改变被操作文件的当前读写位置。命令执行成功返回“真”，失败返回“假”。

参数<2>指定了移动的起始位置。此参数可以为以下常量之一：1（#文件首）；2（#文件尾）；3（#现行位置）。如果本参数被省略，默认值为“#文件首”。

参数<3>指定了从第二个参数的指定位置起移动的距离。如果为正数，即向后移动，如果为负数则向前移动。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 打开内存文件 ()  
移动读写位置 (变量1, #现行位置, 5)  
» 输出调试文本 (取读写位置 (变量1))  
关闭文件 (变量1)

程序运行后将创建一个内存文件并打开；将读写位置从当前位置起向后移动5个字节；在输出面板中显示当前的读写位置；关闭内存文件。

8) “移到文件首” 命令

命令原型为：

<逻辑型> 移到文件首 (整数型 欲进行操作的文件号)

本命令用于改变被操作文件的当前读写位置到文件的头部。命令执行成功返回“真”，失败返回“假”。





例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开内存文件 0

移到文件首 (变量1)

» 输出调试文本 (取读写位置 (变量1))

关闭文件 (变量1)

程序运行后将创建一个内存文件并打开；将读写位置移到文件的头部；在输出面板中显示当前的读写位置；关闭内存文件。

#### 9) “移到文件尾”命令

命令原型为：

<逻辑型> 移到文件尾 (整数型 欲进行操作的文件号)

本命令用于改变被操作文件的当前位置到文件的尾部。命令执行成功返回“真”，失败返回“假”。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开内存文件 0

移到文件尾 (变量1)

» 输出调试文本 (取读写位置 (变量1))

关闭文件 (变量1)

程序运行后将创建一个内存文件并打开它；将读写位置移到文件的尾部；在输出面板中显示当前的读写位置；关闭内存文件。

#### 10) “读入字节集”命令

命令原型为：

<字节集> 读入字节集 (整数型 欲读入数据的文件号, 整数型 欲读入数据的长度)

本命令用于从一个已打开文件中的当前读写位置读入指定长度的字节集数据，实际读入长度（即返回的字节集的尺寸）可能会小于欲读入长度。命令执行成功返回所读出的数据，失败返回空字节集“{}”并且自动将当前文件读写位置移到文件尾部。本命令执行成功后会将读写位置向后移动，移动距离等于读入数据长度。

参数<2>指定了从当前读写位置起读取数据的长度。单位为字节。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	字节集			

文件号 = 打开内存文件 0

变量1 = 读入字节集 (文件号, 3)

» 输出调试文本 (取字节集长度 (变量1))

关闭文件 (文件号)





程序运行后将创建一个内存文件并打开它；从当前读写位置读取3个字节存放在“变量1”中；在输出面板中显示“变量1”中数据的长度；关闭内存文件。

### 11) “写出字节集”命令

命令原型为：

<逻辑型> 写出字节集 (整数型 欲写出数据的文件号, 字节集 欲写出的字节集数据, ... )

本命令用于写出一段或数段字节集数据到已打开文件中的当前读写位置处。该文件被打开时必须拥有写权限。成功返回“真”，失败返回“假”。如果命令执行成功，当前读写位置将向后移动，移动的距离为写出数据的长度。

例如：

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	字节集			

文件号 = 打开内存文件 0  
 写出字节集 (文件号, 到字节集 (“易语言”))  
 移到文件首 (文件号)  
 变量1 = 读入字节集 (文件号, 6)  
 输出调试文本 (取字节集数据 (变量1, #文本型, ))  
 关闭文件 (文件号)

程序运行后将创建一个内存文件并打开它；将“易语言”转换为字节集型数据写到当前读写位置；将当前读写位置移到文件首；读取6个字节的数据存放在“变量1”中；将“变量1”中的数据转换为文本并显示在输出面板中；关闭内存文件。

### 12) “读入文本”命令

命令原型为：

<文本型> 读入文本 (整数型 欲读入文本数据的文件号, [整数型 欲读入文本数据的长度] )

本命令用于在一个已打开的文件中读取指定长度的文本数据，实际读入长度（即返回文本的尺寸）可能会小于欲读入长度。如果该数据中存在字节 0 或 26（文本结束标志），将仅返回该字节之前的数据。命令执行成功返回所读取的文本，失败返回空文本并且自动将当前文件读写位置移到文件尾部。命令成功执行后，会将当前读写位置向后移动，移动距离等于读取文本数据的长度。

参数<2>指定了欲读取的长度。

例如：

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	文本型			





```
文件号 = 打开内存文件 0
» 写出文本 (文件号, “易语言”)
  移到文件首 (文件号)
  变量1 = 读入文本 (文件号, 6)
» 输出调试文本 (变量1)
  关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开它；将“易语言”写到当前读写位置；将当前读写位置移到文件首；读取6个字节的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

### 13) “写出文本”命令

命令原型为：

<逻辑型> 写出文本 (整数型 欲写出文本的文件号, 通用型 欲写出的文本, ...)

本命令用于写出一个或数个文本型数据到已打开文件中的当前读写位置处。该文件被打开时必须拥有写权限。成功返回“真”，失败返回“假”。如果命令执行成功，当前读写位置将向后移动，移动的距离为写出数据的长度。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 0
» 写出文本 (文件号, “易语言”)
  移到文件首 (文件号)
  变量1 = 读入文本 (文件号, 6)
» 输出调试文本 (变量1)
  关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开它；将“易语言”写到当前读写位置；将当前读写位置移到文件首；读取6个字节的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

### 14) “读入一行”命令

命令原型为：

<文本型> 读入一行 (整数型 欲读入文本数据的文件号)

本命令用于从一个已打开的文件中当前读写位置读取并返回一行文本数据，行末的回车及换行符将被抛弃。如果读入失败，将返回一个空文本并且自动将当前文件读写位置移到文件尾部。





例如：

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 ()
» 写出文本 (文件号, “易语言”)
移到文件首 (文件号)
变量1 = 读入一行 (文件号)
» 输出调试文本 (变量1)
关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开它；将“易语言”写到当前读写位置；将当前读写位置移到文件首；读取当前读写位置所在行的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

#### 15) “写文本行”命令

命令原型为：

<逻辑型> 写文本行 (整数型 欲写出文本的文件号, 通用型 欲写出的文本, ...)

本命令用于写出一行或多行文本数据到已打开文件中的当前读写位置处，每行的尾部将被自动加上回车及换行符。该文件被打开时必须拥有写权限。成功返回真，失败返回假。命令执行成功后会将当前读写位置向后移动，移动距离等于写出的数据长度。

例如：

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 ()
» 写文本行 (文件号, “易语言”)
移到文件首 (文件号)
变量1 = 读入一行 (文件号)
» 输出调试文本 (变量1)
关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开它；将“易语言”写到当前读写位置，并自动添加换行；将当前读写位置移到文件首；读取当前读写位置所在行的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

#### 16) “读入数据”命令

命令原型为：

<逻辑型> 读入数据 (整数型 欲读入数据的文件号, 通用型变量/变量数组 存放所读入数据的变量, ...)

本命令应该与“写出数据”命令配合使用，用作从已打开文件中的当前读写位置读取格式数据到指定的一系列变量或数组变量中。成功返回真，失败返回假。命令执行成功







后，当前读写位置会向后移动，移动的距离等于读取数据的长度。

参数<2>指定了读取后数据存放的变量。读出的数据会根据提供变量的类型而自动转换。此参数只支持基本数据类型。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 ()
» 写出数据 (文件号, “易语言”)
  移到文件首 (文件号)
» 读入数据 (文件号, 变量1)
» 输出调试文本 (变量1)
  关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开；将“易语言”写到当前读写位置，并自动添加换行；将当前读写位置移到文件首；读取当前读写位置所在行的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

#### 17) “写出数据”命令

命令原型为：

<逻辑型> 写出数据 (整数型 欲写出数据的文件号, 通用型数组/非数组 欲写出的数据, ...)

本命令应该与“读入数据”命令配合使用，用于写出一系列变量或数组变量的格式数据到文件中当前读写位置处。该文件被打开时必须拥有写权限。成功返回真，失败返回假。命令执行成功后，当前读写位置会向后移动，移动的距离等于写出数据的长度。

具体各种数据类型数据的写出格式如下。

- (1) 数值型、逻辑型、日期时间型、子程序指针型：为其所对应的实际数据。
- (2) 文本型：为文本数据 + 字节 0。
- (3) 字节集型：为字节集数据长度 (整数) + 字节集实际数据。
- (4) 以上各种数据类型的数组型数据：为非数组情况下数据格式的顺序排列。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 ()
» 写出数据 (文件号, “易语言”)
  移到文件首 (文件号)
» 读入数据 (文件号, 变量1)
» 输出调试文本 (变量1)
  关闭文件 (文件号)
```





程序运行后将创建一个内存文件并打开；将“易语言”写到当前读写位置，将当前读写位置移到文件首；读取当前读写位置所在行的数据存放在“变量1”中；在输出面板中显示“变量1”中的内容；关闭内存文件。

18) “是否在文件尾”命令

命令原型为：

<逻辑型> 是否在文件尾 (整数型 文件号, [逻辑型 是否为判断文本已读完])

本命令用于判断一个已打开文件的当前读写位置是否在此文件的最后。如果是则返回“真”，否则返回“假”。

参数<2>指定了是否使用文本结尾做判断。如果为真则将一个文本的结尾作为文件尾。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	逻辑型			

文件号 = 打开内存文件 0  
变量1 = 是否在文件尾 (文件号, 假)  
» 输出调试文本 (变量1)  
关闭文件 (文件号)

程序运行后将创建一个内存文件并打开；获取当前的读写位置并保存在“变量1”中；在输出面板中显示“变量1”的值；关闭内存文件。

19) “取读写位置”命令

命令原型为：

<整数型> 取读写位置 (整数型 文件号)

本命令用于获取一个已打开文件的当前读写位置。读写位置从0开始。命令执行成功返回读写位置，失败返回-1。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 打开内存文件 0  
移到文件首 (变量1)  
» 输出调试文本 (取读写位置 (变量1))  
关闭文件 (变量1)

程序运行后将创建一个内存文件并打开；将读写位置文件移到文件的头部；在输出面板中显示当前的读写位置。关闭内存文件。





## 20) “取文件长度”命令

命令原型为:

&lt;整数型&gt; 取文件长度 (整数型 文件号)

本命令用于获取一个已打开文件的长度, 单位为字节。命令执行成功返回文件长度, 失败返回-1。

例如:

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	整数型			

文件号 = 打开内存文件 ()

变量1 = 取文件长度 (文件号)

» 输出调试文本 (变量1)

关闭文件 (文件号)

程序运行后将创建一个内存文件并打开; 取得此文件的长度并保存在“变量1”中; 在输出面板显示“变量1”的值。关闭内存文件。

## 21) “插入字节集”命令

命令原型为:

&lt;逻辑型&gt; 插入字节集 (整数型 欲写出数据的文件号, 字节集 欲写出的字节集数据, ...)

本命令用于插入一段或数段字节集数据到一个已打开文件中当前读写位置处。该文件被打开时必须拥有写权限。成功返回真, 失败返回假。命令执行成功后, 当前读写位置会向后移动, 移动的距离等于插入数据的长度。

例如:

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	整数型			

文件号 = 打开内存文件 ()

» 插入字节集 (文件号, 到字节集 (“易语言”))

变量1 = 取文件长度 (文件号)

» 输出调试文本 (变量1)

关闭文件 (文件号)

程序运行后将创建一个内存文件并打开; 将“易语言”转换为字节集数据插入到当前读写位置; 取得此文件的长度并保存在“变量1”中; 在输出面板显示“变量1”的值。关闭内存文件。





22) “插入文本” 命令

命令原型为:

<逻辑型> 插入文本 (整数型 欲写出文本的文件号, 通用型 欲写出的文本, ...)

本命令用作插入一个或数个文本数据到文件中当前读写位置处。该文件被打开时必须拥有写权限。成功返回真, 失败返回假。命令执行成功后, 当前读写位置会向后移动, 移动的距离等于插入数据的长度。

例如:

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	文本型			

文件号 = 打开内存文件 0  
 插入文本 (文件号, “易语言”)  
 移到文件首 (文件号)  
 变量1 = 读入一行 (文件号)  
 输出调试文本 (变量1)  
 关闭文件 (文件号)

程序运行后将创建一个内存文件并打开; 将“易语言”插入到当前读写位置; 将当前读写位置移动到文件的头部; 读取一行文本保存在“变量1”中; 在输出面板显示“变量1”的值。关闭内存文件。

23) “插入文本行” 命令

命令原型为:

<逻辑型> 插入文本行 (整数型 欲写出文本的文件号, 通用型 欲写出的文本, ...)

本命令用于插入一个或数个文本数据到文件中当前读写位置处, 并自动添加换行。该文件被打开时必须拥有写权限。成功返回真, 失败返回假。命令执行成功后, 当前读写位置会向后移动, 移动的距离等于插入数据的长度。

例如:

变量名	类 型	静态	数组	备 注
文件号	整数型			
变量1	文本型			

文件号 = 打开内存文件 0  
 插入文本行 (文件号, “易语言”)  
 移到文件首 (文件号)  
 变量1 = 读入一行 (文件号)  
 输出调试文本 (变量1)  
 关闭文件 (文件号)

程序运行后将创建一个内存文件并打开; 将“易语言”插入到当前读写位置; 将当前读写位置移动到文件的头部; 读取一行文本保存在“变量1”中; 在输出面板显示“变量





1”的值。关闭内存文件。

#### 24) “删除数据”命令

命令原型为：

<逻辑型> 删除数据 (整数型 文件号, 整数型 欲删除数据的字节数)

本命令用于从当前读写位置删除其后指定长度的数据。该文件被打开时必须拥有写权限。成功返回真，失败返回假。命令执行成功后会将当前读写位置移动到文件的最后。

例如：

变量名	类型	静态	数组	备注
文件号	整数型			
变量1	文本型			

```
文件号 = 打开内存文件 ()
» 插入文本 (文件号, “易语言”)
移到文件首 (文件号)
删除数据 (文件号, 2)
移到文件首 (文件号)
变量1 = 读入一行 (文件号)
» 输出调试文本 (变量1)
关闭文件 (文件号)
```

程序运行后将创建一个内存文件并打开它；将“易语言”插入到当前读写位置；将当前读写位置移动到文件的头部；删除当前读写位置后的2个字节内容；将当前读写位置移动到文件的头部；读取一行文本保存在“变量1”中；在输出面板显示“变量1”的值。关闭内存文件。

#### 25) “打开加密文件”命令

命令原型为：

<整数型> 打开加密文件 (文本型 欲打开的文件名称, [整数型 打开方式], [整数型 共享方式], [文本型 文件密码], [整数型 明文区长度])

本命令用于打开一个以指定密码加密的文件，以对此文件进行快速安全访问，支持大尺寸文件。成功返回被打开文件的文件号，失败返回0。

参数<2>指定了打开文件的方式。如果省略本参数，默认为“#读写”。参数值可以为以下常量之一：1（#读入：从指定文件读入数据，如果该文件不存在则失败）；2（#写出：写出数据到指定文件，如果该文件不存在则失败）；3（#读写：从文件中读入数据或者写出数据，如果该文件不存在则失败）；4（#重写：写出数据到指定文件。如果该文件不存在则先创建一个新文件，如果已经存在就先清除其中的所有数据）；5（#改写：写出数据到指定文件。如果该文件不存在则创建一个新文件，如果已经存在就直接打开）；6（#改读：从文件中读入数据或者写出数据。如果该文件不存在则创建一个新文件，如果已经存在就直接打开）。





参数<3>指定限制其他进程操作此文件的方式。如果省略本参数，默认为“#无限制”。方式值可以为以下常量之一：1（#无限制：允许其他进程任意读写此文件）；2（#禁止读：禁止其他进程读此文件）；3（#禁止写：禁止其他进程写此文件）；4（#禁止读写：禁止其他进程读写此文件）。

## 3.15 系统处理命令

### 3.15.1 了解剪辑板

剪辑板是指Windows 操作系统提供的一个暂存数据，并且提供共享的一个模块。也称为数据中转站，剪辑板在后台起作用，在内存里，是操作系统设置的一段存储区域，在硬盘中是找不到的。只要在文本输入的地方按CTRL+V或右键“粘贴”剪辑版中的内容就出现了。新的内容送到剪辑板后，将覆盖旧内容。即剪辑板只保存当前的一份内容在内存里，所以，电脑关闭或重启，存在剪辑板中的内容将丢失。

### 3.15.2 了解注册表

注册表（Registry）是Windows中的一个重要的数据库，用于存储系统和应用程序的设置信息。注册表是Windows程序员建造的一个复杂的信息数据库，它是多层次式的。不同Windows版本注册表的基本结构相同。其中的复杂数据会以不同方式上结合，从而产生出一个绝对唯一的注册表。

计算机配置和缺省用户设置的注册表数据在Winnt核心系统中被保存在DEFAULT，SAM，SECURITY，SOFTWARE，SYSTEM，NTUSER.DAT。

注册表由键（或称“项”）、子键（子项）和值项构成。一个键就是分支中的一个文件夹，而子键就是这个文件夹中的子文件夹，子键同样是一个键。一个值项则是一个键的当前定义，由名称、数据类型以及分配的值组成。一个键可以有一个或多个值，每个值的名称各不相同，如果一个值的名称为空，则该值为该键的默认值。

### 3.15.3 系统处理命令

#### 1) “运行”命令

命令原型为：

<逻辑型> 运行（文本型 欲运行的命令行，逻辑型 是否等待程序运行完毕，[整数型 被运行程序窗口显示方式]）

本命令用于运行指定的可执行文件或者外部命令。如果成功，返回真，否则返回假。



参数<1>指定了被运行的可执行文件路径或系统命令。

参数<2>指定了是否等待被运行的程序执行结束再继续执行。“真”为等待被运行程序结束，“假”则不等待。需要注意的是，如为“真”，本程序将进入阻塞状态。

参数<3>指定了被运行程序窗口的显示方式。

该参数可以是以下常量之一：

1（#隐藏窗口）； 2（#普通激活）； 3（#最小化激活）； 4（#最大化激活）； 5（#普通不激活）； 6（#最小化不激活）。如果省略本参数，默认为“普通激活”方式。

例如：

运行（“cmd”，真，#普通激活）

程序运行后会调用“命令行”并等待它结束后再继续执行。

## 2) “取剪辑板文本”命令

命令原型为：

<文本型> 取剪辑板文本 ()

本命令用于获得存放于当前 Windows 系统剪辑板中的文本。如当前剪辑板中无文本数据，将返回空文本。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

置剪辑板文本（“易语言”）

变量1 = 取剪辑板文本 ()

» 输出调试文本 (变量1)

程序运行后会将Windows剪辑版中的内容替换为“易语言”；取得Windows剪辑版中的内容存放在“变量1”中；在输出面板中显示“变量1”中的内容；显示结果为“易语言”。

## 3) “置剪辑板文本”命令

命令原型为：

<逻辑型> 置剪辑板文本 (文本型 准备置入剪辑板的文本)

本命令用于将指定文本存放到目前 Windows 系统剪辑板中去，剪辑板中的原有内容被覆盖。成功返回真，失败返回假。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

置剪辑板文本（“易语言”）

变量1 = 取剪辑板文本 ()

» 输出调试文本 (变量1)





程序运行后会将Windows剪辑版中的内容替换为“易语言”；取得Windows剪辑板中的内容存放在“变量1”中；在输出面板中显示“变量1”中的内容；显示结果为“易语言”。

4) “剪辑板中可有文本” 命令

命令原型为：

<逻辑型> 剪辑板中可有文本 ()

本命令用于判断当前Windows系统剪辑板中是否有文本。如果有文本数据，则返回真，否则返回假。

例如：

变量名	类型	静态	数组	备注
变量1	逻辑型			

变量1 = 剪辑板中可有文本 ()  
» 输出调试文本 (变量1)

程序运行后会检查当前Windows剪辑版中是否有文本，并将结果保存在“变量1”中；在输出面板中显示“变量1”中的内容。

5) “清除剪辑板” 命令

命令原型为：

<无返回值> 清除剪辑板 ()

本命令用于清除当前 Windows 系统剪辑板中的所有数据。

例如：

变量名	类型	静态	数组	备注
变量1	逻辑型			

置剪辑板文本 (“易语言”)  
清除剪辑板 ()  
变量1 = 剪辑板中可有文本 ()  
» 输出调试文本 (变量1)

程序运行后会将Windows剪辑板中的内容替换为“易语言”；清除剪辑板中所有数据；检查当前Windows剪辑板中是否有文本，并将结果保存在“变量1”中；在输出面板中显示“变量1”中的内容。

6) “取屏幕宽度” 命令

命令原型为：

<整数型> 取屏幕宽度 ()

本命令用于获取屏幕当前分辨率的宽度，单位为像素点。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取屏幕宽度 ()

» 输出调试文本 (变量1)

程序运行后会取得系统当前分辨率的宽度保存在“变量1”中；在输出面板中显示“变量1”的值。

#### 7) “取屏幕高度” 命令

命令原型为：

<整数型> 取屏幕高度 ()

本命令用于获取屏幕当前分辨率的高度，单位为像素点。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取屏幕高度 ()

» 输出调试文本 (变量1)

程序运行后会取得系统当前分辨率的高度保存在“变量1”中；在输出面板中显示“变量1”的值。

#### 8) “取鼠标水平位置” 命令

命令原型为：

<整数型> 取鼠标水平位置 ()

本命令用于获取鼠标指针相对于屏幕左边的当前水平位置，单位为像素点。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取鼠标水平位置 ()

» 输出调试文本 (变量1)

程序运行后会取得当前鼠标的水平位置保存在“变量1”中；在输出面板中显示“变量1”的值。

#### 9) “取鼠标垂直位置” 命令

命令原型为：

<整数型> 取鼠标垂直位置 ()

本命令用于获取鼠标指针相对于屏幕顶边的当前垂直位置，单位为像素点。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取鼠标垂直位置 0

» 输出调试文本 (变量1)

程序运行后会取得当前鼠标的垂直位置保存在“变量1”中；在输出面板中显示“变量1”的值。

10) “取颜色数”命令

命令原型为：

<整数型> 取颜色数 ( )

本命令用于获得当前显示方式所提供的最大颜色显示数目。命令执行成功返回所支持的颜色数量，失败返回0。若系统当前颜色数大于16位色深，将返回0。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取颜色数 ( )

» 输出调试文本 (变量1)

程序运行后会取得系统当前屏幕色深所支持的颜色数量并保存在“变量1”中；在输出面板中显示“变量1”的值。

11) “输入框”命令

命令原型为：

<逻辑型> 输入框 ( [文本型 提示信息] , [文本型 窗口标题] , [文本型 初始文本] , 通用型变量 存放输入内容的变量 , [整数型 输入方式] )

本命令用于弹出一个输入对话框供用户输入内容。用户在输入框中点击“确定输入”，返回“真”，以其他方式关闭输入框返回“假”。

**注解：** 本命令会阻塞程序的运行，直到输入框被关闭后程序再继续运行。

参数<1>指定了输入框显示的提示信息。本参数默认为空。

参数<2>指定了输入框的标题。本参数默认为空。

参数<3>指定了初始存在的内容。本参数默认为空。

参数<4>用于取回用户所输入的内容，需要提供第五个参数对应类型的变量。

参数<5>指定了输入方式及类型。本参数可以是以下常量之一：1（#输入文本）；2（#输入整数）；3（#输入小数）；4（#输入密码）。如果省略本参数，默认为“#输入文本”。





例如：

变量名	类型	静态	数组	备注
变量1	文本型			

输入框（“提示信息”，“窗口标题”，“初试文本”，变量1，#输入文本）

» 输出调试文本（变量1）

程序运行后显示一个窗口要求输入信息，将输入的信息保存在“变量1”中；在输出面板中显示“变量1”的内容，如图3-5所示。

## 12) “信息框”命令

命令原型：

<整数型> 信息框（通用型 提示信息，整数型 按钮，[文本型 窗口标题]）

本命令用于弹出一个对话框显示信息，等待用户单击按钮，并返回一个整数告诉程序用户单击哪一个按钮。

该整数为以下常量值之一：0（#确认钮）；1（#取消钮）；2（#放弃钮）；3（#重试钮）；4（#忽略钮）；5（#是钮）；6（#否钮）。

如果对话框有“取消”按钮，则按下ESC键与单击“取消”按钮的效果相同。

**注解：**本命令会阻塞程序的运行，直到对话框被关闭后程序再继续运行。

参数<1>用于指定对话框中显示的内容。本参数可以是任何基础数据类型。

参数<2>指定了对话框中按钮及图标的样式。本参数值由以下几组常量值组成，在将这些常量值相加以生成参数值时，每组值只能取用一个数字（第五组除外）：

第一组（描述对话框中显示按钮的类型与数目）：0（#确认钮）；1（#确认取消钮）；2（#放弃重试忽略钮）；3（#取消是否钮）；4（#是否钮）；5（#重试取消钮）。

第二组（描述图标的样式）：16（#错误图标）；32（#询问图标）；48（#警告图标）；64（#信息图标）。

第三组（说明哪一个按钮是缺省默认值）：0（#默认按钮一）；256（#默认按钮二）；512（#默认按钮三）；768（#默认按钮四）。

第四组（决定如何等待消息框结束）：0（#程序等待）；4096（#系统等待）。

第五组（其他）：65536（#位于前台）；524288（#文本右对齐）。

参数<3>指定了对话框的标题，本参数可以被省略。如果省略，默认为文本“信息：”。

例如：

信息框（“易语言”，#确认钮 + #信息图标，“窗口标题”）

如图3-6所示。

程序运行后会弹出一个对话框并暂停执行程序；关闭对话框后程序继续执行。

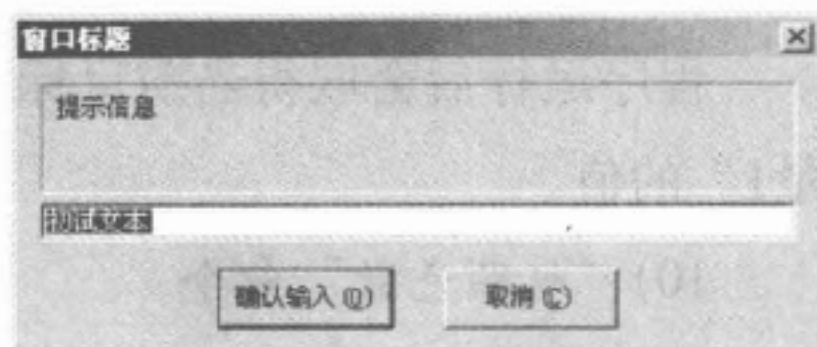


图3-5 窗口标题



图3-6 信息框





### 13) “鸣叫” 命令

命令原型为:

<无返回值> 鸣叫 ()

本命令用于通过计算机媒体设备或者喇叭发出一个声音。

例如:

鸣叫 ()

程序运行后会通过音箱发出一个短促提示音。

### 14) “取启动时间” 命令

命令原型为:

<整数型> 取启动时间 ()

本命令用于获取自 Windows 系统启动后到现在为止所经历过的毫秒数。

例如:

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取启动时间 ()

» 输出调试文本 (变量1)

程序运行后会取得当前时间与系统启动时时间的差，单位为毫秒；并将结果保存在“变量1”中；在输出面板显示“变量1”的值。

### 15) “置等待鼠标” 命令

命令原型为:

<无返回值> 置等待鼠标 ()

本命令用于设置现行鼠标的样式为等待式（一般形状为沙漏形），用作在即将长时间执行程序（阻塞）前提示操作者。作用范围仅为本程序窗口内。当执行完毕后，鼠标样式会恢复为默认指针。晃动鼠标或调用“恢复鼠标”命令即可恢复默认指针。

例如:

子程序名	返回值类型	公开	备 注		
__启动窗口_按下某键	逻辑型				
参数名	类 型	参考	可空	数组	备 注
键代码	整数型				
功能键状态	整数型				

置等待鼠标 ()

延时 (5000)

恢复鼠标 ()

程序执行后将鼠标移动到本程序窗口范围内按下键盘上任意键则鼠标变为等待样式；暂停程序5秒；恢复鼠标样式。





## 16) “恢复鼠标”命令

命令原型为:

&lt;无返回值&gt; 恢复鼠标 ()

本命令用于恢复鼠标状态到默认指针。

例如:

子程序名	返回值类型	公开	备 注		
__启动窗口_按下某键	逻辑型				
参数名	类 型	参考	可空	数组	备 注
键代码	整数型				
功能键状态	整数型				

置等待鼠标 0

延时 (5000)

恢复鼠标 0

程序执行后将鼠标移动到本程序窗口范围内按下键盘上任意键则鼠标变为等待样式; 暂停程序5秒; 恢复鼠标样式。

## 17) “延时”命令

命令原型为:

&lt;无返回值&gt; 延时 (整数型 欲等待的时间)

本命令用于暂停调用本命令所在线程运行并等待指定的时间。时间到后继续程序的执行。

**注解:** 如果使用在主线程中, 会使窗口组件暂时无法处理事件。

例如:

子程序名	返回值类型	公开	备 注		
__启动窗口_按下某键	逻辑型				
参数名	类 型	参考	可空	数组	备 注
键代码	整数型				
功能键状态	整数型				

置等待鼠标 0

延时 (5000)

恢复鼠标 0

程序执行后将鼠标移动到本程序窗口范围内按下键盘上任意键则鼠标变为等待样式; 暂停程序5秒; 恢复鼠标样式。

## 18) “取文本注册项”命令

命令原型为:

&lt;文本型&gt; 取文本注册项 (整数型 根目录, 文本型 全路径注册项名, [文本型 默认文本])

本命令用于在 Windows 注册表中取得指定的文本类型 (REG\_SZ) 注册表项值。命令执行成功返回取得的项值, 失败返回第三个参数设置的文本。





参数<1>指定了欲操作注册表的根目录。

本参数可以是以下常量之一：1（#根类：HKEY\_CLASSES\_ROOT）；2（#现行设置：HKEY\_CURRENT\_CONFIG）；3（#现行用户：HKEY\_CURRENT\_USER）；4（#本地机器：HKEY\_LOCAL\_MACHINE）；5（#所有用户：HKEY\_USERS）。

参数<2>指定了具体欲操作的项名。如欲读取注册项默认值，请在项目名后加“\”，如“test\”。

参数<3>指定了获取失败时返回的文本。默认为空文本。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取文本注册项（#现行用户，“Software\Microsoft\Internet Explorer\Main\Start Page”，“”）

» 输出调试文本（变量1）

程序运行后会读取“HKEY\_CURRENT\_USER\Software\Microsoft\Internet Explorer\Main\Start Page（IE起始页地址）”的值保存在“变量1”中；在输出面板中显示“变量1”中的内容。

#### 19) “取数值注册项”命令

命令原型为：

<整数型> 取数值注册项（整数型 根目录，文本型 全路径注册项名，[整数型 默认数值]）

本命令用于在 Windows 注册表中取得指定的数值类型（REG\_DWORD）注册表项值。命令执行成功返回取得的项值，失败返回第三个参数设置的数值。

参数<1>指定了欲操作注册表的根目录。

本参数可以是以下常量之一：1（#根类：HKEY\_CLASSES\_ROOT）；2（#现行设置：HKEY\_CURRENT\_CONFIG）；3（#现行用户：HKEY\_CURRENT\_USER）；4（#本地机器：HKEY\_LOCAL\_MACHINE）；5（#所有用户：HKEY\_USERS）。

参数<2>指定了具体欲操作的项名。如欲读取注册项默认值，请在项目名后加“\”，如“test\”。

参数<3>指定了获取失败时返回的数值。默认为0。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取数值注册项（#本地机器，“SOFTWARE\Microsoft\Windows\Current Version\Explorer\Advanced\Folder\Hidden\SHOWALL\DefaultValue”，0）

» 输出调试文本（变量1）





程序运行后会读取“HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Folder\Hidden\SHOWALL\DefaultValue（资源浏览器隐藏文件默认设置值）”的值保存在“变量1”中；在输出面板中显示“变量1”中的内容。

## 20) “取字节集注册表项”命令

命令原型为：

<字节集> 取字节集注册项（整数型 根目录，文本型 全路径注册项名，[字节集 默认字节集]）

本命令用于在 Windows 注册表中取得指定的字节集型（REG\_BINARY）注册表项值。命令执行成功返回取得的项值，失败返回第三个参数设置的数值。

参数<1>指定了欲操作注册表的根目录。

本参数可以是以下常量之一：1（#根类：HKEY\_CLASSES\_ROOT）；2（#现行设置：HKEY\_CURRENT\_CONFIG）；3（#现行用户：HKEY\_CURRENT\_USER）；4（#本地机器：HKEY\_LOCAL\_MACHINE）；5（#所有用户：HKEY\_USERS）。

参数<2>指定了具体欲操作的项名。如欲读取注册项默认值，请在项目名后加“\”，如“test\”。

参数<3>指定了获取失败时返回的数值。默认为0。

例如：

变量名	类型	静态	数组	备注
变量1	字节集			

变量1 = 取字节集注册项（#本地机器，“SYSTEM\CurrentControlSet\Services\Winsock\Setup Migration\Providers\Tcpip\WinSock 2.0 Provider ID”，）

→ 输出调试文本（取字节集长度（变量1））

程序运行后会读取“HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Winsock\Setup Migration\Providers\Tcpip\WinSock 2.0 Provider ID（Tcpip提供者ID）”的值保存在“变量1”中；在输出面板中显示“变量1”中内容的长度。

## 21) “写注册项”命令

命令原型为：

<逻辑型> 写注册项（整数型 根目录，文本型 全路径注册项名，通用型 欲写入值）

本命令用于向Windows注册表中保存或建立指定的注册表项。如果欲写项存在，则原有数据将被覆盖。成功返回真，否则返回假。修改注册表有一定危险性，应谨慎行事。

参数<1>指定了欲操作注册表的根目录。

本参数可以是以下常量之一：1（#根类：HKEY\_CLASSES\_ROOT）；2（#现行设置：HKEY\_CURRENT\_CONFIG）；3（#现行用户：HKEY\_CURRENT\_USER）；4（#本地机器：HKEY\_LOCAL\_MACHINE）；5（#所有用户：HKEY\_USERS）。





参数<2>指定了具体欲操作的项名。如欲写注册项默认值，请在项目名后加“\”，如“test\”。

参数<3>指定了欲写入注册表项的值。本参数可以是任意的基本数据类型。

例如：

写注册项 (#现行用户, “Software\Microsoft\Internet Explorer\Main\Start Page”, “http://eyuyan.com”)

程序运行后会将IE的起始页设置为“http://eyuyan.com”。

## 22) “删除表项”命令

命令原型为：

<逻辑型> 删除注册项 (整数型 根目录, 文本型 全路径注册项名)

本命令用于在 Windows 注册表中删除指定注册表项或注册表目录。成功返回真，否则返回假。

**注解：**在删除目录之前必须先删除该目录下所有的项目。修改注册表有一定危险性，应谨慎行事。

参数<1>指定了欲操作注册表的根目录。

本参数可以是以下常量之一：1 (#根类：HKEY\_CLASSES\_ROOT)；2 (#现行设置：HKEY\_CURRENT\_CONFIG)；3 (#现行用户：HKEY\_CURRENT\_USER)；4 (#本地机器：HKEY\_LOCAL\_MACHINE)；5 (#所有用户：HKEY\_USERS)。

参数<2>指定了具体欲操作的项名。如欲删除注册项默认值，请在项目名后加“\”，如“test\”。

例如：

删除注册项 (#现行用户, “Software\Microsoft\Internet Explorer\Main\Start Page”)

信息框 (“点击恢复IE首页”, 0, )

写注册项 (#现行用户, “Software\Microsoft\Internet Explorer\Main\Start Page”, “http://eyuyan.com”)

删除IE起始页的注册表项（如果IE找不到它则将首页默认为微软页面）；弹出信息框暂停程序执行；恢复IE起始页。

## 23) “注册项是否存在”命令

命令原型为：

<逻辑型> 注册项是否存在 (整数型 根目录, 文本型 全路径注册项名)

本命令用于检查指定注册表项是否存在，存在返回真，否则返回假。

参数<1>指定了欲检查注册表的根目录。

本参数可以是以下常量之一：1 (#根类：HKEY\_CLASSES\_ROOT)；2 (#现行设置：HKEY\_CURRENT\_CONFIG)；3 (#现行用户：HKEY\_CURRENT\_USER)；4 (#本地机器：HKEY\_LOCAL\_MACHINE)；5 (#所有用户：HKEY\_USERS)。



参数<2>指定了具体欲检查的项名。如欲检查注册项默认值，请在项目名后加“\”，如“test\”。如果默认值没被设置，则本命令返回假。

例如：

变量名	类 型	静态	数组	备 注
变量1	逻辑型			

变量1 = 注册项是否存在 (#现行用户, "Software\FlySky\E\Install\Path" )

» 输出调试文本 (变量1)

程序运行后会检查“HKEY\_CURRENT\_USER\Software\FlySky\E\Install\ Path（易语言安装路径）”是否存在，并将结果保存在“变量1”中。在输出面板中显示“变量1”的内容。

### 24) “取默认底色”命令

命令原型为：

<整数型> 取默认底色 ( )
-----------------

本命令用于获取Windows系统的默认窗口背景颜色。命令执行成功返回颜色值，失败返回0。改变系统默认颜色可在显示属性—外观—色彩方案中。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = 取默认底色 ( )

» 输出调试文本 (变量1)

程序运行后会取得当前系统默认窗口背景颜色值并保存在“变量1中”。在输出面板中显示“变量1”的内容。

### 25) “快照”命令

命令原型为：

<字节集> 快照 ([整数型 窗口句柄], [整数型 输出宽度], [整数型 输出高度])
-----------------------------------------------

本命令用于捕获指定窗口或屏幕上所有现有显示内容，返回相应图片数据。如果失败，返回空字节集。所取得的数据可直接作为窗口组件的“底图”属性值。

参数<1>指定欲捕获其显示内容的窗口的窗口句柄。如果被省略，默认为捕获屏幕显示内容。

参数<2>指定图片的输出宽度。如果小于0，参数值指定的是最终图片输出宽度相对于所取得图片宽度的百分比（最小为10%）；如果等于0，则按图片原宽度输出；如果大于0，指定输出图片的绝对宽度，单位为像素。如果本参数被省略，默认值为0。

参数<3>指定图片的输出高度。如果小于0，参数值指定的是最终图片输出高度相对于所取得图片高度的百分比（最小为10%）；如果等于0，则按图片原高度输出；如果大于0，指定输出图片的绝对高度，单位为像素。如果本参数被省略，默认值为0。

由于本命令比较耗费资源，不要频繁的使用。



例如：

变量名	类 型	静态	数组	备 注
变量1	字节集			

变量1 = 快照 ( , )  
» 输出调试文本 (取字节集长度 (变量1))

程序运行后会截取当前屏幕的显示内容存储在“变量1”中；在输出面板中显示“变量1”内数据的长度。

### 26) “读配置项”命令

命令原型为：

<文本型> 读配置项 (文本型 配置文件名, 文本型 节名称, 文本型 配置项名称, [文本型 默认文本] )
---------------------------------------------------------

本命令用于读取指定配置文件中指定项目的文本内容。命令执行成功返回配置项的值，失败返回第四个参数所设置的默认值。

参数<1>指定配置文件的名称，通常以.ini作为文件名后缀。本参数可以是一个绝对路径，也可以是程序当前路径的相对路径。

参数<2>包含欲读入配置项所处节的名称。

参数<3>指定欲读入配置项在其节中的名称。

参数<4>如果指定配置项不存在，将返回此默认文本。如果指定配置项不存在且本参数被省略，将返回空文本。

例如：

变量名	类 型	静态	数组	备 注
变量1	文本型			

写配置项 (“config.ini”, “test”, “test”, “易语言”)  
变量1 = 读配置项 (“config.ini”, “test”, “test”, )  
» 输出调试文本 (变量1)

程序运行后会在程序的当前目录中的“config.ini”文件的“test”节点“test”项写入配置项的值为“易语言”；将当前目录中的“config.ini”文件的“test”节点“test”项的值取出并保存在“变量1”中；在输出面板中显示“变量1”的内容。

**注解：** 本命令的第一个参数，如果使用相对路径时应注意以下几点。

(1) 若配置项所在位置为当前目录时，不能直接使用配置项文件名（如“config.ini”）作为第一个参数。若这样做，程序默认是寻找Windows目录中的此配置项文件名；若在配置项文件名前加“\”，如“\config.ini”，此时程序默认寻找的是当前目录的根路径（例当前目录为c:\abc，则程序寻找c:\）。

(2) 若配置项所在位置为当前目录下的子目录时，可以直接使用子目录\配置项文件名（如“abc\config.ini”）的方式来正确的指定它。





## 27) “写配置项”命令

命令原型为:

<逻辑型> 写配置项 (文本型 配置文件名, 文本型 节名称, [文本型 配置项名称], [文本型 欲写入值])

本命令用于修改指定配置文件中指定项目的文本内容。命令执行成功返回真, 失败返回假。

参数<1>指定配置文件的名称, 通常以.ini作为文件名后缀。本参数可以是一个绝对路径, 也可以是程序当前路径的相对路径。如果配置文件不存在, 则自动创建; 但指定的路径必须存在, 否则命令将执行失败。

参数<2>包含欲读入配置项所处节的名称。如果节点不存在, 则自动创建。

参数<3>指定欲读入配置项在其节中的名称。如果不存在, 则自动创建。如果参数值被省略, 则删除指定节及其下的所有配置项。

参数<4>指定了欲写入的配置项值。如果参数值被省略, 则删除所指定配置项。

例如:

变量名	类型	静态	数组	备注
变量1	文本型			

写配置项 (“config.ini”, “test”, “test”, “易语言”)

变量1 = 读配置项 (“config.ini”, “test”, “test”, )

» 输出调试文本 (变量1)

程序运行后会在程序的当前目录中的“config.ini”文件的“test”节点“test”项写入配置项的值为“易语言”; 将当前目录中的“config.ini”文件的“test”节点“test”项的值取出并保存在“变量1”中; 在输出面板中显示“变量1”的内容。

**注解:** 本命令的第一个参数, 如果使用相对路径时应注意以下几点:

(1) 若配置项所在位置为当前目录时, 不能直接使用配置项文件名 (如“config.ini”) 作为第一个参数。若这样做, 程序默认是寻找Windows目录中的此配置项文件名; 若在配置项文件名前加“\”, 如“\config.ini”, 此时程序默认寻找的是当前目录的根路径 (例当前目录为c:\abc, 则程序寻找c:\)。

(2) 若配置项所在位置为当前目录下的子目录时, 可以直接使用子目录\配置项文件名 (如“abc\config.ini”) 的方式来正确的指定它。

## 28) “取配置节点名”命令

命令原型为:

<文本型数组> 取配置节名 (文本型 配置文件名)

本命令用于获取指定配置文件中所有已有节名的文本数组。命令执行成功返回所有节点名的文本数组, 失败返回成员为0的空文本数组。





参数<1>指定配置文件的名称,通常以.ini作为文件名后缀。本参数可以是一个绝对路径,也可以是程序当前路径的相对路径。

例如:

变量名	类型	静态	数组	备注
变量1	文本型		0	
计次变量	整数型			

写配置项 (“config.ini”, “test”, “test”, “易语言”)

变量1 = 取配置节名 (“config.ini”)

--- 计次循环首 (取数组成员数 (变量1), 计次变量)

--- 输出调试文本 (变量1 [计次变量])

--- 计次循环尾 0

程序运行后会在程序的当前目录中的“config.ini”文件的“test”节点“test”项写入配置项的值为“易语言”;获取当前目录中的“config.ini”文件中所有节点名称,并保存在“变量1”中;在输出面板中显示所有节点的名称。

**注解:** 本命令的第一个参数,如果使用相对路径时应注意以下几点:

(1) 若配置项所在位置为当前目录时,不能直接使用配置项文件名(如“config.ini”)作为第一个参数。若这样做,程序默认是寻找Windows目录中的此配置项文件名;若在配置项文件名前加“\”,如“\config.ini”,此时程序默认寻找的是当前目录的根路径(例当前目录为c:\abc,则程序寻找c:\)。

(2) 若配置项所在位置为当前目录下的子目录时,可以直接使用子目录\配置项文件名(如“abc\config.ini”)的方式来正确的指定它。

### 29) “取操作系统类别”命令

命令原型为:

<整数型> 取操作系统类别 ()

本命令用于获取当前操作系统的版本类别。

返回值为以下值之一:

0 (未知); 1 (Windows 32S); 2 (Windows 9X: 包含Win95、Win98、WinME等); 3 (Windows NT: 包含WinNT、Win2000、WinXP等); 4 (Linux)。

例如:

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取操作系统类别 0

--- 输出调试文本 (变量1)

程序运行后会取得当前操作系统类别并保存在“变量1”中;在输出面板显示“变量1”的值。





## 30) “多文件对话框”命令

命令原型为:

<文本型数组> 多文件对话框 ([文本型 标题], [文本型 过滤器], [整数型 初始过滤器], [文本型 初始目录], [逻辑型 不改变目录])

本命令用于显示一个文件打开对话框, 允许用户选择或输入多个所需要打开的已存在文件, 返回用户所选择或输入后的结果文本数组。如果用户未输入或按“取消”按钮退出, 则返回一个成员数为0的空文本数组。

参数<1>指定了文件打开对话框的标题, 如果被省略, 则默认为“请输入欲打开的文件”。

参数<2>指定了过滤器文本。过滤器文本由单个或多个成对的文本串组成, 每对文本串的第一个描述显示形式, 如: “文本文件 (\*.txt)” ; 第二个指定实际的过滤匹配符, 如: “\*.txt”, 所有各文本串之间用“|”号隔开。如果被省略, 则默认没有过滤器(显示所有文件)。

参数<3>指定了初始过滤器索引号。如果上一参数提供了有效的过滤器文本, 则本参数用作指定初始的过滤器, 0为第一个过滤器。如果被省略, 则默认值为0。

参数<4>指定当打开对话框时所自动跳转到的目录, 如果被省略, 则默认为当前目录。

参数<5>指定在对话框关闭后是否自动返回到进入对话框前的文件目录, 如果被省略, 则默认值为假。

例如:

变量名	类型	静态	数组	备注
变量1	文本型		0	
计次变量	整数型			

变量1 = 多文件对话框 (“标题”, “\*. \* (所有文件) | \*. \*”, 0, , )

--- 计次循环首 (取数组成员数 (变量1), 计次变量)  
--- 输出调试文本 (变量1 [计次变量])  
--- 计次循环尾 0

程序运行后会出现一个对话框, 如图3-7所示。如果在对话框中选中文档并点击“打开”按钮, 则在输出面板中显示被选中文件的绝对路径(包括文件名)。

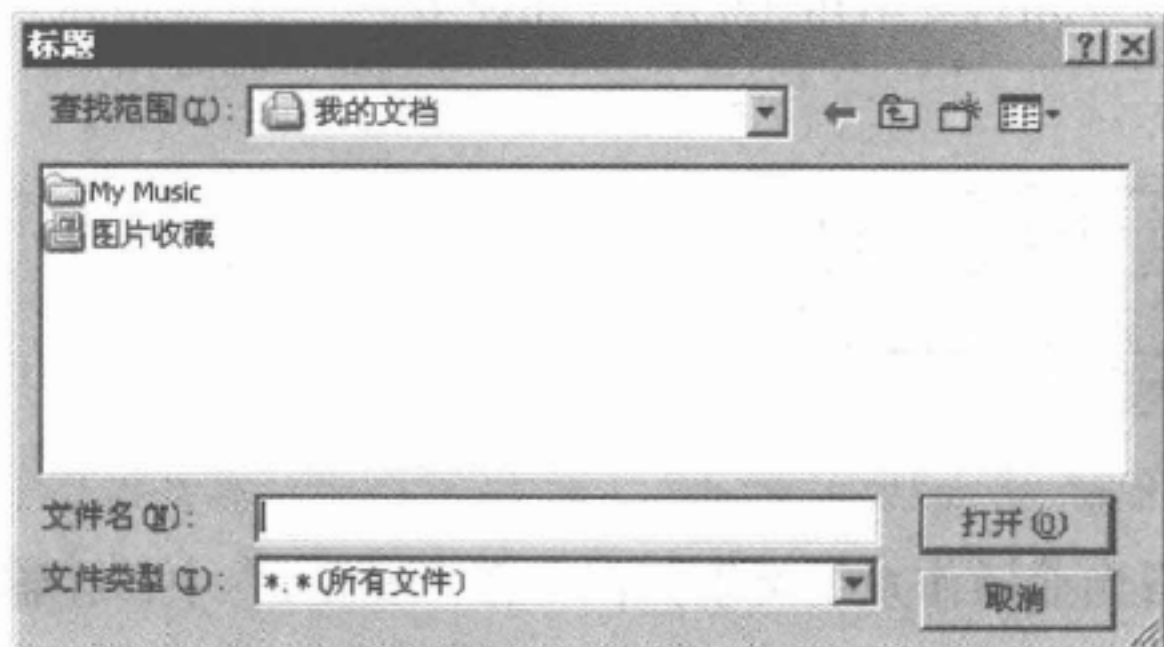


图3-7 选择文件





## 3.16 媒体播放命令

### 3.16.1 常见的音频格式

#### 1) WAV

WAV为微软公司（Microsoft）开发的一种声音文件格式，它符合RIFF（Resource Interchange File Format）文件规范，用于保存Windows平台的音频信息资源，被Windows平台及其应用程序所广泛支持，该格式也支持MSADPCM、CCITT A LAW等多种压缩运算法，支持多种音频数字取样频率和声道，标准格式化的WAV文件和CD格式一样，也是44.1K的取样频率，16位量化数字，因此在声音文件质量上和CD相差无几！

#### 2) MIDI

MIDI（Musical Instrument Digital Interface）乐器数字接口，是20世纪80年代初为解决电声乐器之间的通信问题而提出的。MIDI传输的不是声音信号，而是音符、控制参数等指令，它指示MIDI设备要做什么、怎么做，如演奏哪个音符、多大音量等。它们被统一表示成MIDI消息（MIDI Message）。传输时采用异步串行通信，标准通信波特率为 $31.25 \times (1 \pm 0.01)$  KBaud。

#### 3) MP3

MP3全称是动态影像专家压缩标准音频层面3（Moving Picture Experts Group Audio Layer III）。是当今较流行的一种数字音频编码和有损压缩格式，它设计用来大幅度地降低音频数据量，而对于大多数用户来说重放的音质与最初的不压缩音频相比没有明显的下降。它是在1991年由位于德国埃尔朗根的研究组织Fraunhofer-Gesellschaft的一组工程师发明和标准化的。

### 3.16.2 媒体播放命令

#### 1) “播放音乐”命令

命令原型为：

<逻辑型> 播放音乐（通用型 欲播放的音乐，[逻辑型 是否循环播放]）

本命令用于播放.wav、.mid声音文件或相应格式的字节集声音数据、声音资源。命令执行成功返回“真”，失败返回“假”。

本命令的第一个参数用于指定欲播放的文件名或声音数据。本参数可以是一个文本型，指定了欲播放的文件名；可以是字节集型变量、数据、常量，存放的是声音数据。





例如：

```
播放音乐 ("C:\Windows\Media\ding.wav", 真)
信息框 ("点击停止播放", 0, )
停止播放 0
```

程序执行后会循环播放“C:\Windows\Media\ding.wav”（假设文件存在）；弹出信息框，阻塞程序执行；关闭信息框后停止播放声音。

## 2) “停止播放”命令

命令原型为：

<无返回值> 停止播放 ()

本命令用于停止正在播放的音乐。

本命令对以下命令有效：播放音乐、播放MID、播放MP3、同步播放MP3。

例如：

```
播放音乐 ("C:\Windows\Media\ding.wav", 真)
信息框 ("点击停止播放", 0, )
停止播放 0
```

程序执行后会循环播放“C:\Windows\Media\ding.wav”（假设文件存在）；弹出信息框，阻塞程序执行；关闭信息框后停止播放声音。

## 3) “播放MID”命令

命令原型为：

<无返回值> 播放MID ([整数型 播放次数], [整数型 间隔时间], 通用型数组 / 非数组 欲播放的MIDI音乐, ...)

本命令用于自动连续播放多个 MIDI 声音文件（注意不支持 WAV）或相应格式的字节集声音数据、声音资源。最后一个参数可以扩展多个。

参数<1>指定了播放声音循环的次数；为 -1 表示音乐将被循环播放，否则仅只播放指定的次数。如果本参数被省略，默认值为 1。

参数<2>指定了 MIDI 音乐之间的播放间隔时间，单位为 0.1 秒。如果本参数被省略，默认值为 0。

参数<3>指定了欲播放的MIDI文件或数据。提供参数数据时可以同时提供数组或非数组数据。参数值可以为 MIDI 声音文件名称或相应格式的字节集声音数据、声音资源。

例如：

```
» 播放MID (-1, 0, "C:\WINDOWS\Media\onestop.mid", "C:\WINDOWS\Media\flourish.mid")
信息框 ("点击停止播放", 0, )
停止播放 0
```

程序执行后会循环播放“C:\WINDOWS\Media\onestop.mid”和“C:\WINDOWS\Media\flourish.mid”（假设文件存在）；弹出信息框，阻塞程序执行；关闭信息框后停止





播放声音。

4) “播放MP3” 命令

命令原型为：

<无返回值> 播放MP3 ( [整数型 播放次数] , 文本型数组/非数组 欲播放的MP3 文件名 ; ... )

本命令用于播放一个或自动连续播放多个 MP3 音乐文件。命令参数表中最后一个参数可以被重复添加。

参数<1>指定了播放的循环次数。为 -1 表示指定音乐将被循环播放，否则仅只播放指定的次数。如果本参数被省略，默认值为 1。

参数<2>指定了欲播放的MP3文件的文件名，提供参数数据时可以同时提供数组或非数组数据。

例如：

```
» 播放MP3 (-1, "c:\test.mp3")
   信息框 ("点击停止播放", 0, )
   停止播放 0
```

程序执行后会循环播放“c:\test.mp3”（假设文件存在）；弹出信息框，阻塞程序执行；关闭信息框后停止播放声音。

5) “同步播放MP3” 命令

命令原型为：

<无返回值> 同步播放MP3 ( 文本型 欲播放的MP3文件名 , [整数型 起始播放位置] , [标签 欲通知的反馈标签] , [文本型 保留参数] )

本命令用于播放一次指定的MP3音乐文件，在播放过程中，会自动给指定标签组件发送反馈事件通知，以同步提供当前播放进度百分比。

参数<1>指定了欲播放的MP3文件名。

参数<2>指定了播放起始百分比。如果被省略，默认为0，即从头开始播放。

参数<3>指定了用于获得播放状态的标签。播放时的状态会调用所指定标签的“反馈事件”事件。在播放过程中“反馈事件”的第一个参数将收到当前播放的百分比（0~100）。如果不指定本参数，默认为不反馈播放状态。

参数<4>为保留参数，请保留空。

例如：

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

```
同步播放MP3 ("c:\test.mp3", 0, 标签1, )
信息框 ("点击停止播放", 0, )
停止播放 0
```





子程序名	返回值类型	公开	备 注		
_标签1_反馈事件	整数型				
参数名	类 型	参考	可空	数组	备 注
参数一	整数型				
参数二	整数型				

» 输出调试文本 (参数一)

程序执行后开始播放“c:\test.mp3”（假设文件存在），并将播放时信息反馈到“标签1”中；弹出信息框阻塞程序继续运行；关闭信息框后停止播放。

在“标签1”的反馈事件中将“参数一”的内容显示在输出面板中。显示结果为0~100，即播放的进度百分比。

#### 6) “暂停播放MP3” 命令

命令原型为：

<无返回值> 暂停播放MP3 ()

如果当前正在播放由“播放MP3”或“同步播放MP3”命令所指定的MP3音乐，则将其暂停。

例如：

```
» 播放MP3 (-1, "c:\test.mp3")
  信息框 ("点击暂停播放", 0, )
  暂停播放MP3 ()
  信息框 ("点击继续播放", 0, )
  继续播放MP3 ()
  信息框 ("点击停止播放", 0, )
  停止播放 ()
```

程序执行后开始播放“c:\test.mp3”（假设文件存在）；弹出信息框阻塞程序继续运行；关闭信息框后暂停播放MP3；弹出信息框阻塞程序继续运行；关闭信息框后继续播放MP3；弹出信息框阻塞程序继续运行；关闭信息框后停止播放。

#### 7) “继续播放MP3” 命令

命令原型为：

<无返回值> 继续播放MP3 ()

如果当前由“播放MP3”或“同步播放MP3”命令所播放MP3音乐已经被暂停，则继续播放。

例如：

```
» 播放MP3 (-1, "c:\test.mp3")
  信息框 ("点击暂停播放", 0, )
  暂停播放MP3 ()
  信息框 ("点击继续播放", 0, )
  继续播放MP3 ()
  信息框 ("点击停止播放", 0, )
  停止播放 ()
```





程序执行后开始播放“c:\test.mp3”（假设文件存在）；弹出信息框阻塞程序继续运行；关闭信息框后暂停播放MP3；弹出信息框阻塞程序继续运行；关闭信息框后继续播放MP3；弹出信息框阻塞程序继续运行；关闭信息框后停止播放。

#### 8) “取MP3播放状态”命令

命令原型为：

<整数型> 取MP3播放状态 ()

本命令用于获得当前由“播放MP3”或“同步播放MP3”命令所播放MP3音乐的播放状态：0（已经被停止或未播放）；-1（正在播放但被暂停）；1（正在播放）。

例如：

```
» 播放MP3 (-1, "c:\test.mp3")
» 输出调试文本 (取MP3播放状态 ()
  信息框 ("点击停止播放", 0, )
  停止播放 ()
```

程序执行后会循环播放“c:\test.mp3”（假设文件存在）；在输出面板中显示当前播放状态，显示结果为1，即播放中；弹出信息框，阻塞程序执行；关闭信息框后停止播放声音。

## 3.17 网络通信命令

### 3.17.1 了解IP地址

所谓IP地址就是给每个连接在Internet上的主机分配的一个32bit地址。按照TCP/IP协议规定，IP地址用二进制来表示，每个IP地址长32bit，比特换算成字节，就是4个字节。例如一个采用二进制形式的IP地址是“00001010000000000000000000000001”，这么长的地址，人们处理起来也太费劲了。为了方便人们的使用，IP地址经常被写成十进制的形式，中间使用符号“.”分开不同的字节。于是，上面的IP地址可以表示为“10.0.0.1”。IP地址的这种表示法叫做“点分十进制表示法”，这显然比1和0容易记忆得多。

### 3.17.2 了解域名与主机名

域名（Domain Name），是由一串用点分隔的名字组成的互联网上某一台计算机或计算机组的名称，用于在数据传输时标识计算机的电子方位（有时也指地理位置）。域名的出现解决了IP地址不易记忆的弊病，通过域名可以转换为对应的实际IP地址。

主机名就是计算机的名字，通常在局域网内通信时作用与域名相同。





这个名字可以随时改，从“我的电脑”的“属性”里有个“计算机名”，更改即可。用户登录的时候用的是操作系统的个人用户账号，这个也可以更改，从控制面板的用户界面里改就可以了。这个用户名和计算机名无关。

### 3.17.3 易语言中的网络通信命令

易语言使用的网络通信命令有以下几条。

#### 1) “取主机名”命令

命令原型为：

<文本型> 取主机名 ()

本命令用于返回主机的名称，用作在网络通信中标志本机地址。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 取主机名 ()

» 输出调试文本 (变量1)

程序执行后获取本机的主机名并保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 2) “通信测试”命令

命令原型为：

<整数型> 通信测试 (文本型 被测试主机地址, [整数型 最长等待时间])

本命令用于测试与指定主机是否能够正常通信。返回被测试主机的通信响应时间。如果无法通信或者测试失败，返回 -1。

参数<1>指定了被测试的主机地址。主机地址可以是一个IP、域名或主机名。

参数<2>指定了最长等待时间，单位为毫秒。超过此时间即认为无法与被测试主机通信。如果省略本参数，则默认为  $10 \times 1000$  毫秒，即 10 秒。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 通信测试 (“www.eyuyan.com”, )

» 输出调试文本 (变量1)

程序执行后对本机到“www.eyuyan.com”进行通信测试，并将结果保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 3) “转换为主机名”命令

命令原型为：





<文本型> 转换为主机名 (文本型 欲转换IP地址)

本命令用于将IP地址转换为对应的主机名或域名。命令执行成功返回转换后的结果，失败返回空文本。本命令在对广域网转换时需要DNS服务器的支持（反查）。

参数<1>指定了欲转换的IP地址。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 转换为主机名 (“127.0.0.1”)

» 输出调试文本 (变量1)

程序执行后获取“127.0.0.1”对应的主机名并保存在“变量1”中；在输出面板中显示“变量1”的内容。

4) “转换为IP地址”命令

命令原型为：

<文本型> 转换为IP地址 (文本型 欲转换主机名)

本命令用于通过主机名或域名获取对应的IP地址。命令执行成功返回转换后的结果，失败返回空文本。

参数<1>指定了欲转换的主机名或域名。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 转换为IP地址 (“www.eyuyan.com”)

» 输出调试文本 (变量1)

程序执行后获取“www.eyuyan.com”所对应的IP地址并保存在“变量1中”；在输出面板中显示“变量1”的内容。

3.18 其他命令

控制台操作命令只在程序类型是“Windows控制台程序”或“Linux控制台程序”时有效。但易语言所有控制台程序均是32位程序，不能在16位环境（如DOS）下运行。

1) “标准输出”命令

命令原型为：

<无返回值> 标准输出 ([整数型 输出方向], 通用型 欲输出内容, ...)





本命令用于在标准输出设备或标准错误设备上输出指定的内容。命令参数表中最后一个参数可以被重复添加。

参数<1>指定了输出目标设备。本参数提供内容所输出到的设备。

可以为以下常量值之一：1（#标准输出设备）；2（#标准错误设备）。如果省略本参数，默认为“#标准输出设备”。

参数<2>指定了在输出设备上显示的内容。本参数只能为文本、数值、逻辑值或日期时间。如果内容为文本且包含多行，可在各行之间用回车符（即“字符（13）”）、换行符（即“字符（10）”）或回车换行符的组合（即：“字符（13）+字符（10）”）来分隔。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

» 标准输出（#标准输出设备，“易语言”+ #换行符）  
 变量1 = 标准输入（真）  
 » 输出调试文本（变量1）

程序运行后会在系统的标准输出设备上显示“易语言”并换行；等待用户输入内容，用户确认输入后程序继续执行；在输出面板中显示用户所输入的内容。

## 2) “标准输入”命令

命令原型为：

<文本型> 标准输入（[逻辑型 是否回显]）

本命令用于在标准输入设备上请求输入最多包含2048个字符的一行文本，用户确认输入后返回用户所输入的内容。

参数<1>决定输入时是否显示所输入字符，为假不显示，为真显示。如果被省略，默认值为真，即回显。可以通过将本参数设置为假以输入密码等特殊信息。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

» 标准输出（#标准输出设备，“易语言”+ #换行符）  
 变量1 = 标准输入（真）  
 » 输出调试文本（变量1）

程序运行后会在系统的标准输出设备上显示“易语言”并换行；等待用户输入内容，用户确认输入后程序继续执行；在输出面板中显示用户所输入的内容。

## 3) “载入”命令

命令原型为：

<逻辑型> 载入（窗口欲载入的窗口，[窗口父窗口]，逻辑型 是否采用对话框方式）





本命令用于加载指定的窗口，根据窗口的“可视”属性决定是否显示它。成功返回真，失败返回假。使用时应注意，不要载入一个已加载的窗口，可使用“是否已创建”命令判断窗口是否已被加载。

参数<1>指定了欲载入的窗口。

参数<2>指定欲载入窗口的父窗口，此窗口必须已经被载入。如果本参数被省略，表示无父窗口。如果窗口具有父窗口，其位置将永远位于父窗口的上层；当父窗口被关闭时，窗口将自动被关闭；当父窗口被禁止时，窗口将自动被禁止。

参数<3>指定用户在对被载入窗口进行操作的同时是否允许同时对其他窗口进行操作，如果本参数值为真且被载入窗口的“可视”属性为真，则命令将一直等待到该窗口被销毁后才返回。本参数默认为“真”。

例如：

载入(窗口1,,假)

程序执行后会载入“窗口1”（假设窗口1存在）。

#### 4) “选择”命令

命令原型为：

<通用型> 选择 (逻辑型 用作选择的逻辑值, 通用型 待选择项一, 通用型 待选择项二)

本命令用于根据所提供参数的值，返回两部分中的其中一个。使用时需要注意返回值的类型是根据所返回的参数而改变。

参数<1>指定了返回哪个参数。如果该值为真，将返回第一个待选择项（第二个参数），否则将返回第二个待选择项（即第三个参数）。

参数<2>指定了返回选项一，当第一个参数为真时返回本参数。

参数<3>指定了返回选项二，当第一个参数为假时返回本参数。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 选择 (1 + 1 = 2, “2”, “不等2”)

» 输出调试文本 (变量1)

程序运行会判断“1+1”是否等于“2”，若等于则返回第一个值“2”，否则返回第二个值“不等2”，将结果保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 5) “多项选择”命令

命令原型为：

<通用型> 多项选择 (整数型 索引值, 通用型 待选择项数据, ...)





本命令用于从参数列表项目中选择并返回一个值。命令参数表中最后一个参数可以被重复添加。使用时需要注意返回值的类型是根据所返回的参数而改变。

参数<1>指定了欲返回参数的索引值。如果索引值是 1，则命令会返回列表中的第 1 个选择项。如果索引值是 2，则会返回列表中的第 2 个选择项，以此类推。如果索引值小于 1 或者大于最大可选择项，将会产生运行错误。

参数<2>及后续扩展指定了一个返回选项。

例如：

变量名	类型	静态	数组	备注
变量1	文本型			

变量1 = 多项选择 (1 + 1, “一”, “二”, “三”)

» 输出调试文本 (变量1)

程序执行后会根据多项选择的第一个参数选择一个值保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 6) “是否为空”命令

命令原型为：

<逻辑型> 是否为空 (通用型变量/变量数组 欲测试的参数变量)

本命令用于指出在调用子程序时是否为指定参数提供了数据 (该参数必定有“可空”标志)，如未提供，返回真，否则返回假。

参数<1>指定了欲测试的子程序参数。提供参数数据时只能提供变量及变量数组。

例如：

子程序名	返回值类型	公开	备 注		
子程序1					
参数名	类 型	参考	可空	数组	备 注
参数1	文本型		✓		

变量名	类型	静态	数组	备注
变量1	逻辑型			

变量1 = 是否为空 (参数1)

» 输出调试文本 (变量1)

“子程序1”被调用后程序会对它的“参数1”进行检查，调用时是否没有提供具体参数值，并将测试结果保存在“变量1”中；在输出面板中显示“变量1”的值。

#### 7) “是否已创建”命令

命令原型为：

<逻辑型> 是否已创建 (通用型 欲被检查的窗口或窗口组件)

本命令用于检查一个窗口或窗口组件是否已被创建。如果指定的窗口或窗口组件有效且已经被载入或创建，则返回真，否则返回假。对于窗口菜单项组件，如果其所处窗口已





经被载入，命令将返回真，否则返回假。

参数<1>指定欲测试的目标。

例如：

变量名	类 型	静态	数组	备 注
变量1	逻辑型			
变量1 = 是否已创建 (窗口1)				
» 输出调试文本 (变量1)				

程序运行后会检查“窗口1”是否已创建，并将结果保存在“变量1”中；在输出面板中显示“变量1”的值。

8) “取数据类型尺寸” 命令

命令原型为：

<整数型> 取数据类型尺寸 (整数型 欲取其尺寸的数据类型)

本命令用于获取指定基础定长数据类型的数据尺寸，该数据类型不能为文本、字节集、库定义类型或用户自定义数据类型。程序执行成功返回对应数据类型所占内存的长度，单位字节；失败返回0。

参数<1>指定了欲取其尺寸的数据类型。

本参数值可以为以下常量之一：1（#字节型）；2（#短整数型）；3（#整数型）；4（#长整数型）；5（#小数型）；6（#双精度小数型）；7（#逻辑型）；8（#日期时间型）；9（#子程序指针型）。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			
变量1 = 取数据类型尺寸 (#子程序指针型)				
» 输出调试文本 (变量1)				

程序执行后会获取“子程序指针型”所占内存空间的长度，并保存在“变量1”中；在输出面板中显示“变量1”的内容。

9) “取颜色值” 命令

命令原型为：

<整数型> 取颜色值 (整数型 红色，整数型 绿色，整数型 蓝色)

本命令用于返回一个整数，用来表示一个红绿蓝光组合而成的颜色值。

参数<1>指定了红色光在所生成的颜色中所占的比重。数值范围从 0 到 255。

参数<2>指定了绿色光在所生成的颜色中所占的比重。数值范围从 0 到 255。

参数<3>指定了蓝色光在所生成的颜色中所占的比重。数值范围从 0 到 255。





例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取颜色值 (255, 25, 66)

» 输出调试文本 (变量1)

程序运行后会取得一个由三种光以一定比例组合而成的颜色值，并将它保存在“变量1”中；在输出面板中显示“变量1”的值。

#### 10) “取事件组件”命令

命令原型为：

<通用型> 取事件组件 ()

本命令用于返回一个有效的组件型数据，指定当前所正在处理的事件是从哪个组件实际产生的。本命令主要用于在事件处理子程序中动态确定事件的来源。但只能在处理事件的过程首部使用本命令，如果在其他情况下使用，将产生运行时错误。

例如：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
新按钮	按钮		

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

复制窗口组件 (按钮1, 新按钮)

新按钮.左边 = 按钮1.左边 + 按钮1.宽度

新按钮.可视 = 真

事件转移 (新按钮, 按钮1)

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	按钮			

变量1 = 取事件组件 ()

--- 判断 (变量1 = 按钮1)

» 输出调试文本 (“按钮1被单击”)

--- 判断 (变量1 = 新按钮)

» 输出调试文本 (“新按钮被单击”)

本程序演示了如何动态生成一个新组件及对新组件的属性操作和响应事件。

本例程源码见随书光盘中“\图书例程\第三章\其他\事件转移.e”。





## 11) “事件转移”命令

命令原型为:

<逻辑型> 事件转移 (通用型 组件一, [通用型 组件二])

本命令用于设置将第一个组件上产生的所有事件转交由第二个同窗口同类型组件的事件处理子程序去处理, 就好象此事件是在第二个组件上实际发生的一样。本命令可用来设置用同一事件处理子程序处理来自多个同类型组件的事件。使用时可与“取事件组件”命令、对象的“标记”属性等相配合。设置成功返回真, 失败返回假。

参数<1>指定欲将其事件转移的组件。如果该参数指定的不是窗口内的组件, 而是窗口本身, 则不考虑“组件二”参数的设置情况, 直接取消被指定窗口上所有组件事件转移设置。

参数<2>指定“组件一”事件转移到的目的组件, 必须与“组件一”的类型相同且在同一窗口内, 否则命令会失败。如果本参数被省略, 将取消“组件一”的原有事件转移设置。

例如:

窗口程序集名	保 留	保 留	备 注
窗口程序集1			
变量名	类 型	数 组	备 注
新按钮	按钮		

子程序名	返回值类型	公 开	备 注
_启动窗口_创建完毕			

复制窗口组件 (按钮1, 新按钮)

新按钮.左边 = 按钮1.左边 + 按钮1.宽度

新按钮.可视 = 真

事件转移 (新按钮, 按钮1)

子程序名	返回值类型	公 开	备 注
_按钮1_被单击			

变量名	类 型	静 态	数 组	备 注
变量1	按钮			

变量1 = 取事件组件 ()

判断 (变量1 = 按钮1)

→ 输出调试文本 (“按钮1被单击”)

判断 (变量1 = 新按钮)

→ 输出调试文本 (“新按钮被单击”)

本程序演示了如何动态生成一个新组件及对新组件的属性操作和响应事件。

本例程源码见随书光盘中 “\图书例程\第三章\其他\事件转移.e”

## 12) “复制窗口组件”命令

命令原型为:





<逻辑型> 复制窗口组件 (通用型 欲被复制的窗口组件, 通用型变量 存放新创建组件的变量)

本命令用于复制创建与指定窗口组件设计状态完全一致的新组件。该被复制组件将放在原组件的下面, 其事件被自动转移到原组件上。如果复制时被复制窗口组件所在的窗口尚未被载入, 那么复制出来的窗口组件也不会被立即创建, 而会等到其所在窗口将来被载入时一并创建。并且此时复制出来的窗口组件将永久存在, 就好象是用户在设计窗口时所加入的一样, 不会因为其所在窗口被销毁而被抛弃; 如果复制时被复制窗口组件所在的窗口已经被载入, 该组件会被立即创建, 但此时无论该组件的“可视”属性是真还是假, 创建后的单元都将不会被显示, 以便于用户进行属性调整。此时复制出来的窗口组件为临时存在, 在其所在窗口被销毁后, 此窗口组件将被抛弃, 下次再载入原窗口时此窗口组件将不复存在。成功返回真, 失败返回假。

参数<1>指定了欲复制的组件原型。参数值不能为窗口本身、窗口菜单项组件、选择夹窗口组件。

参数<2>指定了复制出的新组件所存的变量。提供参数数据时只能提供变量。该变量数据类型必须与被复制窗口组件数据类型完全一致, 否则命令会失败。

例如:

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
新按钮	按钮		

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

复制窗口组件 (按钮1, 新按钮)

新按钮. 左边 = 按钮1. 左边 + 按钮1. 宽度

新按钮. 可视 = 真

事件转移 (新按钮, 按钮1)

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	按钮			

变量1 = 取事件组件 0

```

判断 (变量1 = 按钮1)
├── 输出调试文本 ("按钮1被单击")
└── 判断 (变量1 = 新按钮)
    ├── 输出调试文本 ("新按钮被单击")
    └──

```

本程序演示了如何动态生成一个新组件及对新组件的属性操作和响应事件。

本例程源码见随书光盘中 “\图书例程\第三章\其他\事件转移.e”





13) “处理事件” 命令

命令原型为:

<无返回值> 处理事件 ()

本命令用于暂时转让控制权，以便让 Windows 操作系统有机会处理其他的如用户键盘或鼠标输入等事件。直到操作系统处理并发送完程序队列中的所有事件后，命令才会返回。本命令一般用于在进行大量运算时不使窗口组件长时间失去响应。

例如:

子程序名	返回值类型	公开	备注
按钮1_被单击			

```
--> 判断循环首 (是否已创建 (启动窗口))
    处理事件 ()
    延时 (1000)
--> 判断循环尾 ()
```

程序运行后，当点击“按钮1”，程序即进入一个死循环（除非“\_启动窗口”被销毁即程序结束）至程序结束；在循环中调用“处理事件”命令来让程序强制处理未处理的事件；使用延时的目的是为了模拟耗时很长的运算。

**注解：**若修改此程序，不使用“处理事件”命令，那么新程序在点击“按钮1”后将进入假死状态（不能处理事件）。

本例程源码见随书光盘中“\图书例程\第三章\其他\处理事件.e”

14) “载入图片” 命令

命令原型为:

<整数型> 载入图片 (通用型 欲被载入的图片)

本命令用于载入指定图片，以供画板、打印机等对象的“画图片”、“取图片宽度”、“取图片高度”之类成员命令使用。如成功则返回被载入图片的图片号，失败返回0。在不使用已载入的图片时应使用“卸载图片”命令以释放资源。

参数<1>指定了欲被载入的图片。

参数值可以为JPG、GIF、BMP、DIB、ICO、CUR 等被支持格式的图片数据、图片资源常量或图片文件名。

例如:

变量名	类型	静态	数组	备注
图片号	整数型			

```
图片号 = 载入图片 ("C:\WINDOWS\Prairie Wind.bmp")
输出调试文本 (图片号)
卸载图片 (图片号)
```

程序运行后会载入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在），并将图片



号保存在变量中；在输出面板中显示“图片号”的内容；卸载已载入的图片，释放资源。

### 15) “卸载图片”命令

命令原型为：

<无返回值> 卸载图片 (整数型 图片号)

本命令用于卸载被载入的图片，图片被卸载后不能再被使用。如果某图片被载入后未被卸载，将在易程序退出时将自动被卸载。

例如：

变量名	类型	静态	数组	备注
图片号	整数型			

图片号 = 载入图片 (“C:\WINDOWS\Prairie Wind.bmp”)

» 输出调试文本 (图片号)

卸载图片 (图片号)

程序运行后会载入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在），并将图片号保存在变量中；在输出面板中显示“图片号”的内容；卸载已载入的图片，释放资源。

### 16) “取硬盘特征字”命令

命令原型为：

<整数型> 取硬盘特征字 ()

本命令用于返回电脑中第一个物理硬盘的物理特征字，该特征字是仅与硬件相关的，也就是说与任何软件系统都无关（包括操作系统）。用户可以使用此特征字来限制自己的程序仅在某一台计算机上运行，以保护自己软件的版权。本命令可以在任何 Windows 系统版本下运行。命令执行后如果返回 0，表示此次取硬盘特征字失败。由于有可能是因为暂时的 I/O 冲突造成，因此失败后可以等待一段随机时间后再试。如果重复尝试四五次后仍然失败，表明该硬盘无法取出特征字。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取硬盘特征字 ()

» 输出调试文本 (变量1)

程序运行取得硬盘特征码并保存在“变量1”中；在输出面板中显示“变量1”的内容。

### 17) “取系统语言”命令

命令原型为：

<整数型> 取系统语言 ()

本命令用于获取当前系统运行时环境所支持的语言版本类型；





返回值为以下常量值之一：1（GBK中文）；2（英文）；3（BIG5中文）；4（日文）。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 取系统语言 ()

» 输出调试文本 (变量1)

程序运行后取得当前程序所在系统的语言版本类型，并保存在“变量1”中；在输出面板中显示“变量1”的内容。

### 18) “写到内存”命令

命令原型为：

<无返回值> 写到内存 (通用型数组/非数组 欲写到内存的数据, 整数型 内存区域指针, [整数型 内存区域尺寸])

本命令用于将数据写到指定的内存区域，注意调用本命令前一定要确保所提供的内存区域真实有效，且写出的数据不超过范围。本命令的最佳使用场合就是在易语言回调子程序和易语言DLL公开子程序中用作对外输出数据。

参数<1>指定了欲写到内存的数据，提供参数数据时可以同时提供数组或非数组数据。参数值只能为基本数据类型数据或字节数组。

参数<2>指定了欲写向内存区域首地址的指针值。

参数<3>指定了该内存区域的有效尺寸，如果欲写出数据超出此尺寸值，将被自动切除。参数值如果为-1，则表示易语言不对欲写的数据做长度处理，但在使用时要确保操作不越界（关于内存越界请查看相关资料）。如果本参数被省略，则默认值为-1。

例如：

变量名	类型	静态	数组	备注
内存地址	整数型			

内存地址 = VirtualAlloc (0, 1024 × 1024 × 10, #MEM\_COMMIT, #PAGE\_READWRITE)

写到内存 (“易语言”, 内存地址, -1)

» 输出调试文本 (指针到文本 (内存地址))

VirtualFree (内存地址, 10 × 1024 × 1024, #MEM\_DECOMMIT)

程序运行后会申请10M内存；将“易语言”写入内存；将内存中的数据以文本型读出；释放内存。

本例程源码见随书光盘中“\图书例程\第三章\其他\写到内存.e”

### 19) “DLL命令调用转向”命令

命令原型为：

<逻辑型> DLL命令调用转向 ([文本型 DLL库文件名], [文本型 调用命令名])





本命令用于设置下次程序中执行任意一条DLL命令时所使用的DLL库文件名和在DLL库文件中的调用命令名，此设置将覆盖该被调用DLL命令在程序设计时所提供的原始值。但要注意本命令仅对一次调用有效，即在程序执行一次任一DLL命令后，本命令的设置值将自动取消，不再对后续DLL命令调用起作用。可以通过本命令来实现对非固定DLL库文件中的DLL命令调用。命令执行成功返回真，如果指定DLL库文件不存在或该库文件中不存在所指定的调用命令名，则返回假。

**注解：**转向的目标应和被转向的构造相同（返回值及参数个数和类型）。

参数<1>指定了欲调用DLL命令所处的DLL库全路径文件名，如果本参数为空文本或被省略，则取消上次调用本命令时所设置的信息，使其不再影响程序中对DLL命令的调用。

参数<2>指定欲调用DLL命令在其所处DLL库文件中的名称。如果本参数为空文本或被省略，则当程序中下次调用DLL命令时将使用其程序设计时所提供的原始命令名。最好为本参数提供一个有效的调用命令名，这样做可以事先检查该调用命令名是否存在。

例如：

```
MessageBoxA (0, "易语言", "标题", 0)
DLL命令调用转向 ("MyDLL.dll", "MessageBoxA")
MessageBoxA (0, "易语言", "标题", 0)
```

程序运行后会调用MessageBoxA（这时调用的是系统API）显示一个信息框；将MessageBoxA的调用转向到“MyDLL.dll”（我们编写的）中的同名命令；再次调用MessageBoxA（这时调用的是我们编写的DLL中的命令）。

本例程源码见随书光盘中“\图书例程\第三章\其他\DLL调用转向.rar”。

## 20) “置错误提示管理”命令

命令原型为：

<无返回值> 置错误提示管理（【子程序指针 用作进行错误提示的子程序】）

本命令用于设置当运行时如果产生了导致程序崩溃的严重错误时用来对该错误进行详细提示的子程序地址，如果未进行此项设置，发生严重错误时运行时环境将自动提示该错误的详细信息并直接退出。

参数<1>指定了用作进行错误提示的子程序。注意该子程序必须接收两个参数，第一个参数为整数型，用作接收错误代码。

参数<2>为文本型，用作接收详细错误文本。同时该子程序必须返回一个逻辑值，返回真表示已经自行处理完毕，系统将不再显示该错误信息，返回假表示由系统来继续显示该错误信息。以上设置必须完全正确，否则结果不能预测。

另外，还应注意以下几点。

(1) 该错误提示子程序不要再引发新的错误。

(2) 在进入该错误提示子程序后，系统将自动关闭事件消息通知处理机制，也就是说





任何事件将无法得到响应。

(3) 无论如何，当该子程序调用退出后，系统将自动将整个应用程序关闭。

如果省略本参数，系统将恢复错误提示的默认处理方式。

例如：

子程序名	返回值类型	公开	备 注
__置错误提示管理按钮_被单击			

变量名	类 型	静态	数组	备 注
数组	整数型		0	

加入成员 (数组, 1)

置错误提示管理 (&置错误提示)

信息框 (数组 [2], 0, )

子程序名	返回值类型	公开	备 注		
置错误提示	逻辑型				
参数名	类 型	参考	可空	数组	备 注
错误代码	整数型				
错误文本	文本型				

--- 如果真 (错误代码 = 1)

  信息框 (错误文本, 0, )

返回 (真)

本例程源码见随书光盘中 “\图书例程\第三章\其他\置错误提示管理.e”

## 21) “置DLL装载目录” 命令

命令原型为：

<文本型> 置DLL装载目录 ( [文本型 DLL装载目录] )

本命令用于设置当程序中执行到DLL命令时装载其DLL库文件的优先装载路径，即系统将优先到该路径下去装载指定的DLL文件。本命令所设置结果对所有DLL命令设置中的未指定全路径的DLL库文件装载均有影响，且在程序运行期间全程有效。命令执行后返回系统在本次设置以前的值。

**注解：**应在第一次DLL调用前执行此命令，否则命令无效。

参数<1>指定系统对DLL命令配置中DLL库文件的优先装载路径，如果被省略，则默认值为空文本。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_创建完毕			

置DLL装载目录 (“a\”)

MessageBoxA (0, “易语言”, “标题”, 0)

本例程源码见随书光盘中 “\图书例程\第三章\其他\置DLL装载目录.rar”





## 22) “取组件名称” 命令

命令原型为:

&lt;文本型&gt; 取组件名称 (通用型 窗口组件)

本命令用于返回指定窗口组件的名称, 如果所提供对象不是窗口组件, 将返回空文本。本命令一般用于在动态生成窗口组件后使用。

参数&lt;1&gt;指定了欲取名称的窗口组件。

例如:

输出调试文本 (取组件名称 (按钮1))

程序执行后将在输出面板中显示“按钮1”的组件名称。

## 23) “取对象类型” 命令

命令原型为:

&lt;文本型&gt; 取对象类型 (通用型 欲取类型的对象)

本命令用于取得指定对象在支持库内对应的库类型名称。如果指定对象的类型为在支持库内定义的数据类型, 则返回该数据类型的名称, 否则返回空文本。

参数&lt;1&gt;指定了欲取类型的对象, 如图3-8所示。

例如:

输出调试文本 (取对象类型 (按钮1))

程序运行后会取得“按钮1”所对应的数据类型名称, 并显示在输出面板中。显示结果为“按钮”。

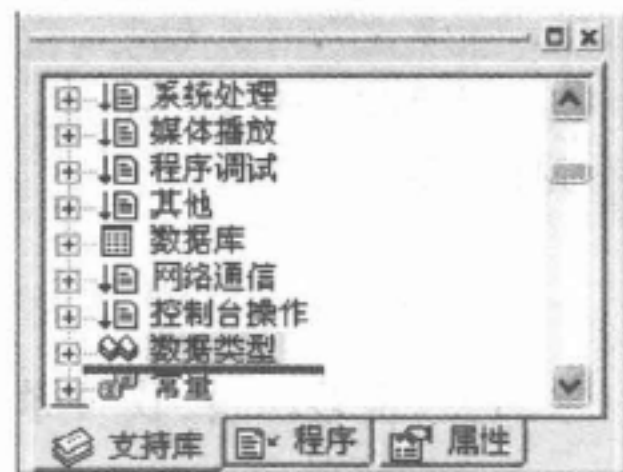


图3-8 取对象类型

## 24) “寻找组件” 命令

命令原型为:

&lt;整数型&gt; 寻找组件 (通用型 父组件, [文本型 名称前缀], [文本型 类型], [整数型 标记最小值], [整数型 标记最大值])

本命令用于按照指定条件寻找符合要求的窗口组件, 成功返回非零目标句柄, 未找到返回零。若不指定寻找条件则返回容器中所有组件。

**注解:** 所返回的非零目标句柄在不再使用后必须调用“清除组件寻找句柄”命令释放。

参数&lt;1&gt;指定欲寻找组件的直接或间接父组件, 可以为容器组件或窗口。

参数<2>指定欲寻找组件的名称前缀, 只有以此名称开头的组件才能符合要求。如果本参数被省略, 默认为空文本, 即不做判断。

参数<3>指定欲寻找组件的类型名称, 如果提供空文本, 则表示无类型要求。如果本参数被省略, 默认为空文本。

参数<4>指定欲寻找组件的标记的最小值, 只有其“标记”属性转换到的数值大于或等于该值的组件才能符合要求。如果本参数被省略, 默认为最小可能整数值。

参数&lt;5&gt;指定欲寻找组件的标记的最大值, 只有其“标记”属性转换到的数值小于





或等于该值的组件才能符合要求。如果本参数被省略，默认为最大可能整数值。

例如：

变量名	类型	静态	数组	备注
组件句柄	整数型			
数量	整数型			
按钮组	按钮		0	
计次变量	整数型			

组件句柄 = 寻找组件 (启动窗口, , “按钮”, , )

数量 = 取找到组件数目 (组件句柄)

» 输出调试文本 (数量)

» 重定义数组 (按钮组, 假, 数量)

--- 计次循环首 (数量, 计次变量)

按钮组 [计次变量] = 取所找到组件 (组件句柄, 计次变量 - 1)

» 输出调试文本 (取组件名称 (按钮组 [计次变量]))

--- 计次循环尾 0

清除组件寻找句柄 (组件句柄)

程序运行后寻找“\_启动窗口”中的所有类型为按钮的组件，并将结果保存在“组件句柄”中；获取寻找结果的数量；在输出面板中显示结果数量；改变“按钮组”的成员数为结果数；使用循环将所有结果取出存入“按钮组”中，并在输出面板中显示每个结果的名称；清除句柄，释放资源。

本例程源码见随书光盘中“\图书例程\第三章\其他\寻找组件.e”

## 25) “取找到组件数目”命令

命令原型为：

<整数型> 取找到组件数目 (整数型 组件寻找句柄)

本命令用于返回“寻找组件”命令所找到满足条件的组件数目。

参数<1>指定了由“寻找组件”命令所返回的寻找句柄。

例如：

变量名	类型	静态	数组	备注
组件句柄	整数型			
数量	整数型			
按钮组	按钮		0	
计次变量	整数型			

组件句柄 = 寻找组件 (启动窗口, , “按钮”, , )

数量 = 取找到组件数目 (组件句柄)

» 输出调试文本 (数量)

» 重定义数组 (按钮组, 假, 数量)

--- 计次循环首 (数量, 计次变量)

按钮组 [计次变量] = 取所找到组件 (组件句柄, 计次变量 - 1)

» 输出调试文本 (取组件名称 (按钮组 [计次变量]))

--- 计次循环尾 0

清除组件寻找句柄 (组件句柄)



程序运行后寻找“\_启动窗口”中的所有类型为按钮的组件，并将结果保存在“组件句柄”中；获取寻找结果的数量；在输出面板中显示结果数量；改变“按钮组”的成员数为结果数；使用循环将所有结果取出存入“按钮组”中，并在输出面板中显示每个结果的名称；清除句柄，释放资源。

本例程源码见随书光盘中“\图书例程\第三章\其他\寻找组件.e”

### 26) “取所找到的组件”命令

命令原型为：

<通用型> 取所找到组件 (整数型 组件寻找句柄, 整数型 组件索引位置)

本命令用于返回“寻找组件”命令所找到的指定索引位置处的组件。

参数<1>指定由“寻找组件”命令所返回的寻找句柄。

参数<2>指定了欲取回的结果索引，索引从0开始。本参数必须大于等于零且小于“取找到组件数目”所返回的寻找到组件数目。

例如：

变量名	类 型	静态	数组	备 注
组件句柄	整数型			
数量	整数型			
按钮组	按钮		0	
计次变量	整数型			

```

组件句柄 = 寻找组件 (启动窗口, , “按钮”, , )
数量 = 取找到组件数目 (组件句柄)
» 输出调试文本 (数量)
» 重定义数组 (按钮组, 假, 数量)
    -- 计次循环首 (数量, 计次变量)
        按钮组 [计次变量] = 取所找到组件 (组件句柄, 计次变量 - 1)
    » 输出调试文本 (取组件名称 (按钮组 [计次变量]))
    --- 计次循环尾 0
清除组件寻找句柄 (组件句柄)
    
```

程序运行后寻找“\_启动窗口”中的所有类型为按钮的组件，并将结果保存在“组件句柄”中；获取寻找结果的数量；在输出面板中显示结果数量；改变“按钮组”的成员数为结果数；使用循环将所有结果取出存入“按钮组”中，并在输出面板中显示每个结果的名称；清除句柄，释放资源。

本例程源码见随书光盘中“\图书例程\第三章\其他\寻找组件.e”

### 27) “置入代码”命令

命令原型为：

<无返回值> 置入代码 (通用型 代码数据)

本命令用于在编译后文件代码段中当前语句位置置入指定的机器指令码数据。



参数<1>指定了欲置入到代码段中的机器指令码数据（二进制机器码），可以是字节集数据或二进制文件名文本。需要注意的是本参数不支持变量。

例如：

变量名	类型	静态	数组	备注
模块句柄	整数型			
函数入口地址	子程序指针			

```
' 模块句柄 = LoadLibrary ("user32.dll")
' 函数入口地址 = GetProcAddress (模块句柄, "MessageBoxA")
置入代码 ({ 184, 136, 5, 213, 119, 106, 0, 106, 0, 106, 0, 106, 0, 255, 208 })
' FreeLibrary (模块句柄)
' 对应汇编代码为
' mov EAX 函数入口地址
' push 0
' push 0
' push 0
' push 0
' call EAX
```

通过置入代码调用“MessageBoxA”函数。

本例程源码见随书光盘中“\图书例程\第三章\其他\置入代码.e”，请在Windows2000或以上系统执行本例程。

### 3.19 我的播放器（三）

在第三章学习了易语言核心支持库中的命令部分。其中就有媒体播放类命令，可以持续循环的播放MP3。在本章节，根据学习到的易语言的命令，来对MP3播放器加以升级。使“我的播放器”可以循环播放多个MP3文件，并增加暂停播放和继续播放功能。

程序中需要实现三个主要功能：循环播放多个MP3文件；暂停播放；继续播放。那么用三个按钮来对应完成此三项功能，可以分三步来编写代码。

第一步：设计界面。

新建一个易语言程序，在启动窗口中加入三个按钮，分别设置按钮组件的属性，并将“启动窗口”的标题改为“我的播放器（三）”。

按钮1标题设置为：播放，按钮1的名称设置为：播放按钮；用作加入MP3文件，并循环播放MP3文件；

按钮2标题设置为：暂停，按钮2的名称设置为：暂停按钮；用作暂停播放MP3文件；

按钮3标题设置为：继续，按钮3的名称设置为：继续按钮；用作继续播放MP3文件。

调整各组件的大小和位置，界面效果如图3-9所示。



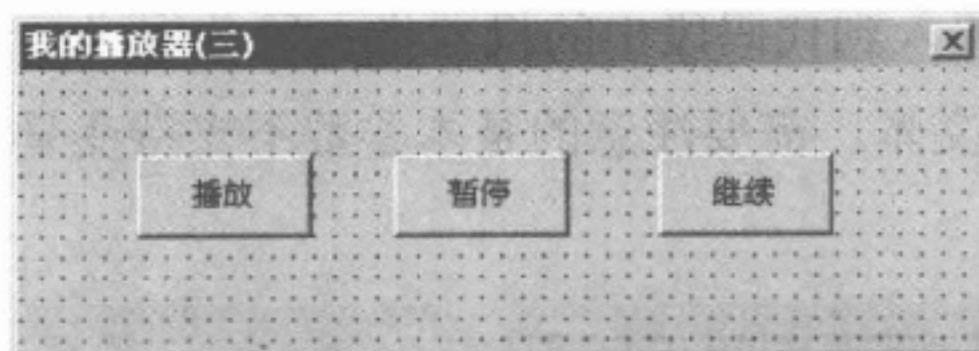


图3-9 界面效果

第二步：编写循环播放多个MP3文件的代码。

要播放MP3，先要将MP3文件找到并将MP3文件名加入到程序中。在系统处理类命令中有一个“多文件对话框”命令可以实现打开目录选择MP3文件的功能，将选择的MP3文件存放到MP3文件组变量中。双击“播放按钮”编写如下代码：

子程序名	返回值类型	公开	备注
_播放按钮_被单击			

变量名	类型	静态	数组	备注
MP3文件组	文本型		0	

MP3文件组 = 多文件对话框 (“添加mp3文件”, “mp3文件 (\*.mp3) | \*.mp3”, , , 真)

--- 如果真 (取数组成员数 (MP3文件组) = 0)

返回 ()

→ 播放MP3 (-1, MP3文件组)

在“多文件对话框”命令中，选择MP3文件，返回结果文本数组。如果未输入或按“取消”按钮退出，没有选择MP3文件，则返回一个成员数为0的空文本数组。所以，在播放MP3文件前要先判断一下数组的成员数，没有选择MP3文件就返回。

这样就可以实现循环播放MP3文件了。

思考：怎样知道每次播放的是哪个MP3文件？

第三步：编写暂停播放和继续播放的代码。

在媒体播放类命令中有播放MP3的命令，也有暂停播放和继续播放的命令。双击“暂停按钮”和“继续按钮”，实现暂停播放和继续播放代码如下：

子程序名	返回值类型	公开	备注
_暂停按钮_被单击			

暂停播放MP3 ()

子程序名	返回值类型	公开	备注
_继续按钮_被单击			

继续播放MP3 ()

至此，使用“我的播放器”循环播放多个MP3文件的程序编写完成了。

思考：每次播放MP3文件，都要重新载入MP3文件，那么怎样能够保存MP3文件播放列表？在程序启动时又怎样调用MP3文件播放列表呢？





### 3.20 小结

将解决问题的一段程序的封装既为命令。命令的参数用于传递解决问题所需要的数据；命令的返回值是解决问题得到的结果。

易语言中命令的调用格式为：[返回值] [所属对象].命令名 ([参数1], [参数2], ...)，易语言中的命令参数和返回值可以是数组形式的。易语言核心库中提供了很多常用的基础命令。

### 3.21 习题

3-1 什么是命令？

3-2 易语言中命令的调用格式为[返回值] \_\_\_\_\_ . \_\_\_\_\_ ([参数1], [参数2], ...)。

3-3 在循环内，使用什么命令可以结束本次循环，进入下次循环？

3-4 使用什么命令可以获得当前的系统时间？

3-5 使用什么命令可以将其他数据类型的数据转换到文本型？







## 第四章 易语言组件

### 本章目标

在本章结束时，我们能够：

- 了解什么是窗口和窗口组件
- 掌握易语言中的窗口和窗口组件共有的事件、属性、方法
- 掌握易语言中菜单的创建、编辑、使用
- 掌握易语言中基本组件的使用



在Windows应用程序中常会发现一些诸如按钮、编辑框、选择框、列表框等，这些都被称为组件。在程序开发中经常要用到类似的组件，而组件是一种特殊样式的窗口，所以又叫窗口组件且跟窗口拥有很多相同的元素。一个Windows窗口程序一般都要包含若干个不同的组件，相互配合分别实现不同的功能。如果将开发的程序比喻成组装一辆汽车，那么组件就好比是构成汽车的“零件”；而将这些“零件”组装成一辆车的过程，其实就是为组件编写代码和组件集成的过程。

窗口组件是程序界面可视化操作的基本单元，也是程序中主要的输入、输出部件。所有的组件都是一个独立的对象，拥有自己的属性、事件和命令。在易语言中有很多窗口组件，这些组件涉及软件开发中各方面的应用。下文详细介绍易语言核心支持库中组件的使用。

## 4.1 窗口

窗口是窗口组件、菜单最基础的容器，是Windows窗口程序最明显的特征。

### 4.1.1 窗口的定义

窗口是Windows程序中最基础的显示组件，几乎Windows中所有的可视内容都基于窗口。窗口组件也可以看作是一个依附在窗口中拥有特殊样式的窗口，所以窗口与窗口组件有许多共有事件、属性、命令。

易语言在默认设置情况下，每新建一个易程序窗口程序，都会自动对应生成一个“\_启动窗口”，“\_启动窗口”中的代码是易语言Windows窗口程序最先执行的。窗口有自己的属性、事件和命令，并且可以作为其他组件的载体和组件。在易语言中新增窗口可使用菜单或在程序面板中使用鼠标右键添加，如图4-1所示。

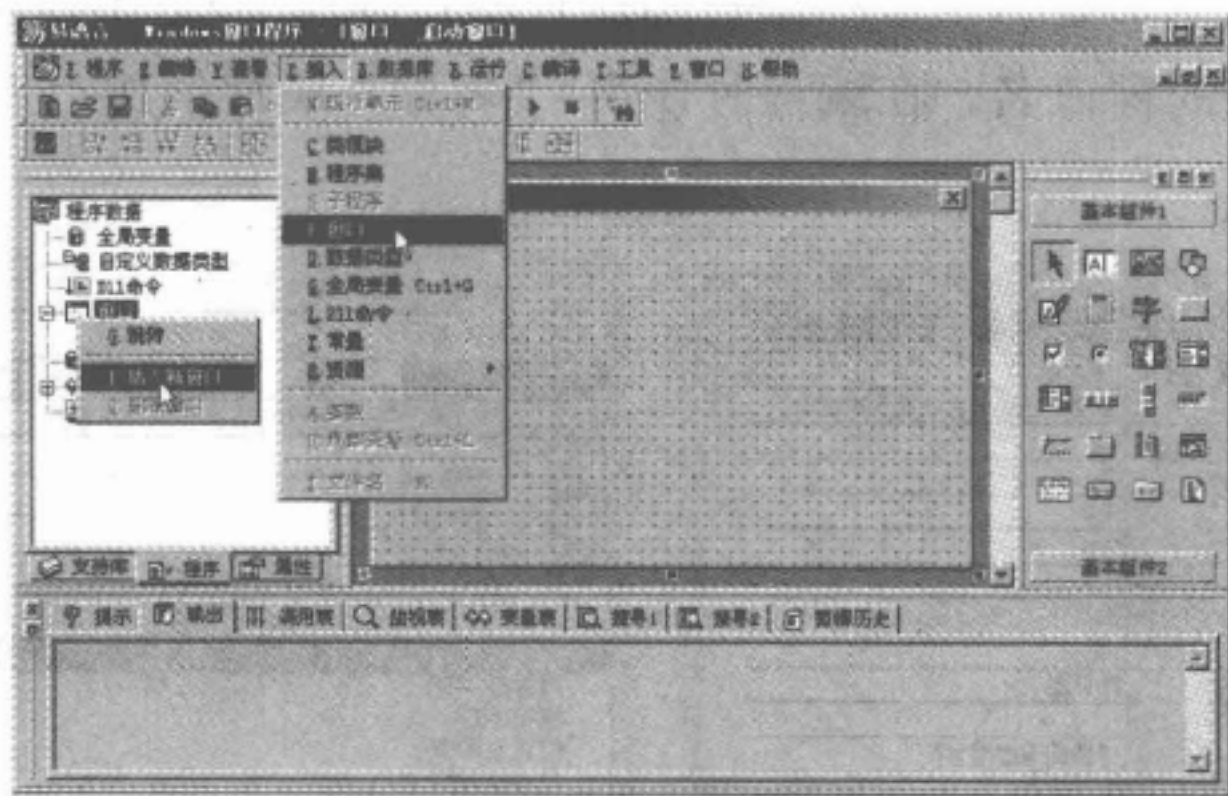


图4-1 新建窗口

### 4.1.2 共有属性

属性：用于表示窗口或窗口组件的功能性质及与其他对象之间的关系。



共有属性：即所有窗口和窗口组件共同具有的基本属性。如：“左边”、“顶边”、“宽度”、“高度”、“标记”、“可视”、“禁止”和“鼠标指针”等。

虽然所有窗口组件都拥有共有属性，但表现在具体窗口组件中的效果可能会有所不同。另外某些共有属性在特殊窗口组件上没有意义。在程序设计时可以通过选中窗口组件，来查看此对象的有效属性，如图4-2所示。

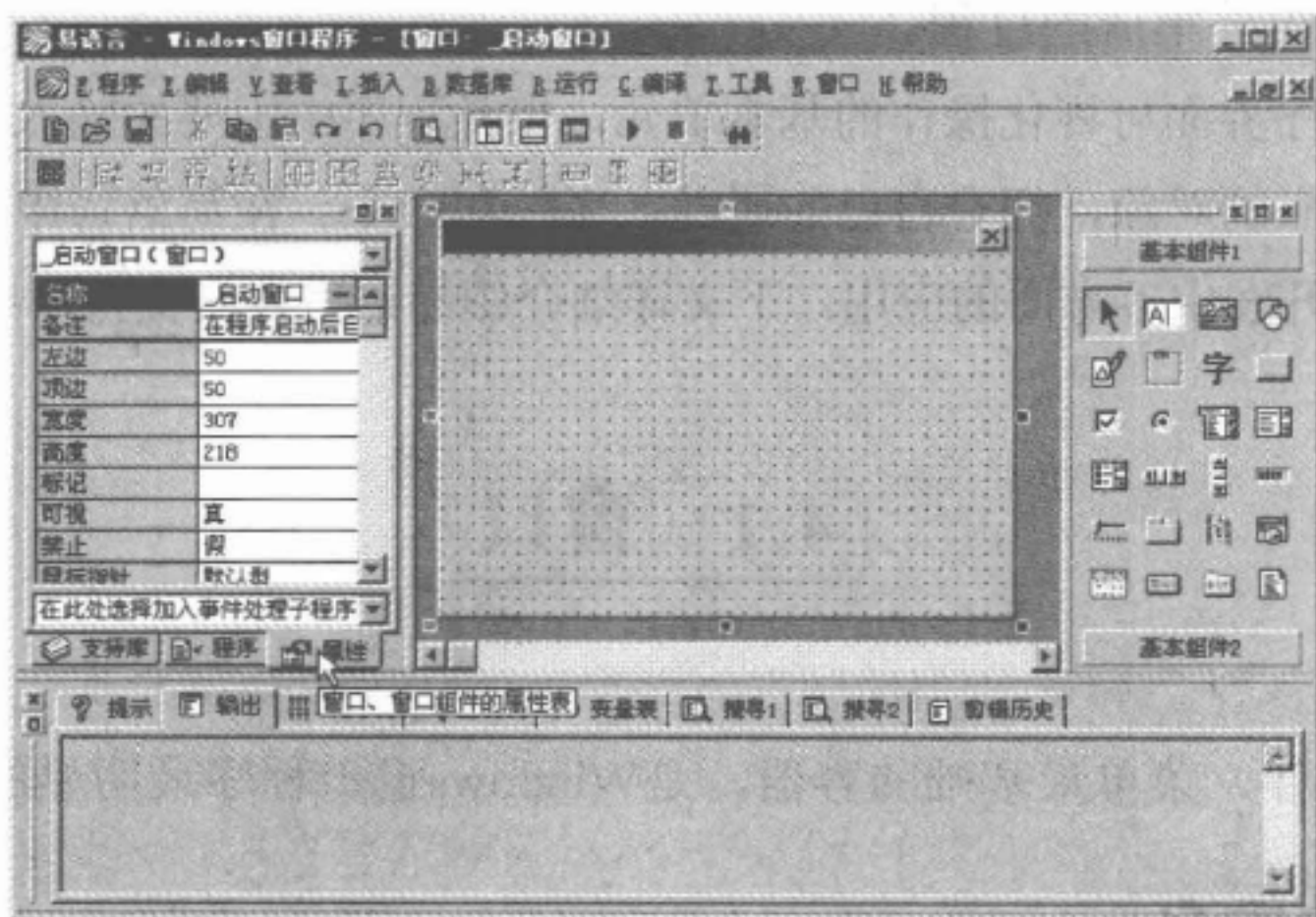


图4-2 窗口、窗口组件的属性表

图4-2中，左侧属性窗口中，组件下拉列表中为被选择的对象名称，事件列表上面为该对象的有效属性名，及对应的当前属性值。在设计程序时只要激活待设置窗口或窗口组件，点击属性面板，修改待调整属性栏的属性值即可。在程序设计时指定的属性将作为程序运行后的初始样式。

#### 1) “名称”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用

本属性指定了在当前窗口中唯一的窗口组件名，如图4-3所示。如对象是窗口，则代表在整个程序中唯一的窗口名。在编程时使用“名称”属性值加上“.”就可以调用对应对象的命令和属性了。

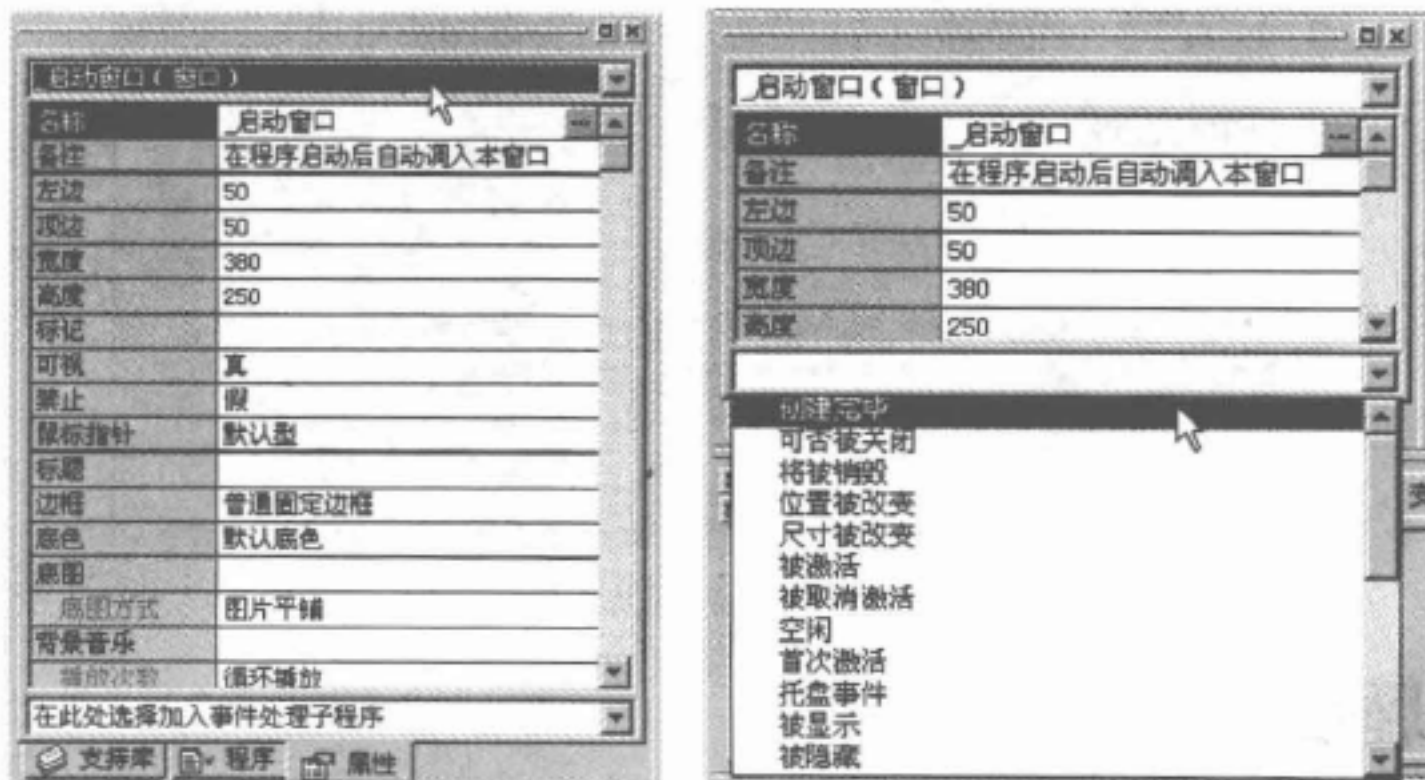


图4-3 \_启动窗口事件处理子程序



例如：

在属性面板选择加入事件处理子程序中选择“创建完毕”事件，切换到代码编辑界面。

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

▶ 输出调试文本 (启动窗口.宽度)  
启动窗口.高度 = 300

程序运行后会在输出面板中显示“\_启动窗口”当前的宽度属性值，并改变“\_启动窗口”的高度属性为300像素。

2) “备注”属性

属性类型：文本型；有效范围：设计时；  
编程时权限：不可用

本属性用于记录有关当前所选择对象的备注信息。本属性不对窗口及窗口组件外观、功能、事件产生任何影响，只用作在设计时方便记录提示信息，如图4-4所示。类似于代码编写时的注释信息。

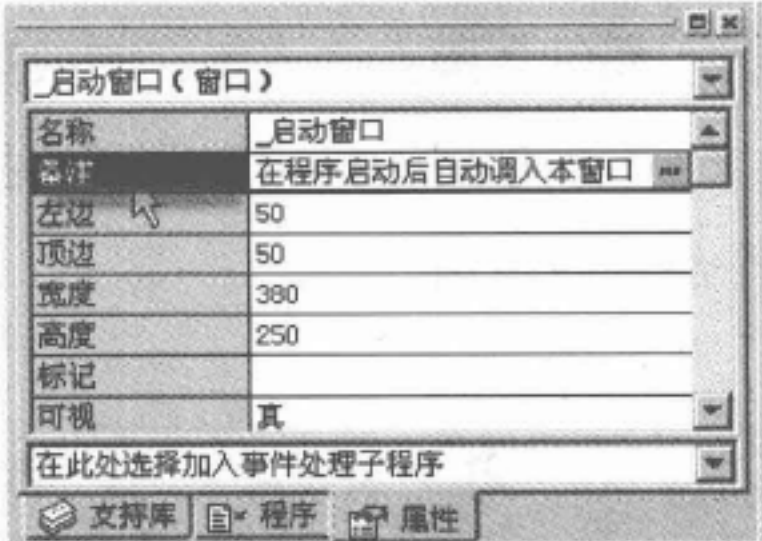


图4-4 窗口备注

3) “左边”属性

属性类型：整数型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于记录当前窗口左上角相对于屏幕左边的位置，或当前窗口组件在父窗口中的相对位置。单位为像素点。改变本属性后，对象的位置即发生改变。

例如：

输出调试文本 (启动窗口.左边)

程序运行后会在输出面板中显示“\_启动窗口”左上角相对于屏幕左边的相对位置。

4) “顶边”属性

属性类型：整数型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于记录当前窗口左上角相对于屏幕顶边的位置，或当前窗口组件在父窗口中的相对位置。单位为像素点。改变本属性后，对象的位置即发生改变。

例如：

输出调试文本 (启动窗口.顶边)

程序运行后会在输出面板中显示“\_启动窗口”左上角相对于屏幕顶边的相对位置。

5) “宽度”属性

属性类型：整数型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于记录当前窗口或窗口组件的宽度。单位为像素点。





例如：

```
_启动窗口.宽度 = 400
```

程序运行后会改变“\_启动窗口”的宽度为400像素。

#### 6) “高度”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于记录当前窗口或窗口组件的高度。单位为像素点。

例如：

```
_启动窗口.高度 = 300
```

程序运行后会改变“\_启动窗口”的高度为300像素。

#### 7) “标记”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于标记当前所选择对象的附加文本数据。本属性一般用于简化编程时控制多个组件时使用。

例如：

```
_启动窗口.标记 = “易语言”
```

输出调试文本(\_启动窗口.标记)

程序运行后会改变“\_启动窗口”的标记属性值为“易语言”；在输出面板中显示“\_启动窗口”的标记属性值。

#### 8) “可视”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置对象是否显示可见。“真”为可见，“假”为不可见。需要注意的是，为“假”时实际对象仍然存在，所占用的资源并没有被释放。若要释放不使用的窗口或窗口组件所占用的资源应调用“销毁”命令。

例如：

```
按钮_可视.可视 = 假
```

程序运行后会将“按钮\_可视”的可视属性设为“假”使“按钮\_可视”不可见。

#### 9) “禁止”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置对象是否禁止用户操作。“真”为不可操作，“假”为可操作。

例如：

```
输出调试文本(_启动窗口.禁止)
```

程序运行后在输出面板中显示当前“\_启动窗口”的禁止属性值。





10) “鼠标指针” 属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置当鼠标移动到本组件上时鼠标指针的形状。本属性只在该对象“可视”为“真”且“禁止”为“假”时有效。如果欲在程序中动态设置本属性为内置值，应该使用字节集数据。如：“鼠标指针 = 到字节集(4)”语句可以将鼠标指针设置为沙漏型。如果欲使用外部鼠标指针，则必须保证其格式为“cur”。

内置值有：0（默认型）；1（标准箭头型）；2（十字型）；3（文本编辑型）；4（沙漏型）；5（箭头及问号型）；6（箭头及沙漏型）；7（禁止符型）；8（四向箭头型）；9（北<->东箭头型）；10（北<->南箭头型）；11（北<->西箭头型）；12（西<->东箭头型）；13（向上箭头型）；14（手型）。

例如：

变量名	类型	静态	数组	备注
指针数据	字节集			

```
指针数据 = 读入文件 ("C:\WINDOWS\Cursors\3dgarro.cur")
_启动窗口.鼠标指针 = 指针数据
```

程序运行后会读取指针数据并存放在“指针数据”变量中；设置“\_启动窗口”的鼠标指针属性为“指针数据”所存放的内容（例中假设文件存在）。

11) “可停留焦点” 属性

**属性类型：**逻辑型；**有效范围：**设计时；

本属性用于设置当用户使用 TAB键或光标键在各组件之间移动焦点时是否允许在本组件上停留。“真”为允许停留，“假”为不停留。

**注：**本属性只影响焦点切换时是否允许停留，不会影响组件主动拥有焦点（如使用“获取焦点”命令等方式）。

12) “停留顺序” 属性

**属性类型：**整数型；**有效范围：**设计时；

当“可停留焦点”属性为“真”时，本属性用于定义用户使用 TAB键或光标键在各组件之间移动焦点时的顺序。在同父组件中，停留顺序越小，移动焦点时顺序越前。

例如：

在窗口上添加两个编辑框和一个颜色选择器，如图4-5所示。

将“编辑框2”的“可停留焦点”属性设为“假”；将“编辑框1”的“停留顺序”属性设为“1”，如图4-6所示。

程序执行后，焦点停留在“颜色选择器1”上；使

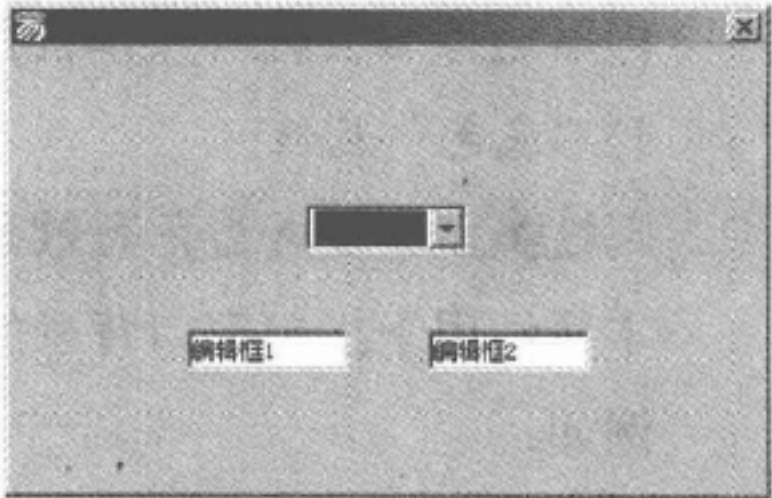


图4-5 焦点切换演示





用TAB键切换焦点时可以发现，焦点始终在“编辑框1”与“颜色选择器1”之间切换。



图4-6 停留顺序设置

### 4.1.3 独有属性

窗口除了拥有共有属性外，还拥有一些独有的属性，这些属性称为独有属性。

#### 1) “标题”属性

**属性类型：**文本型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
本属性用于指定窗口的标题文字。

例如：

```
_启动窗口.标题 = “易语言”
```

程序执行后会将“\_启动窗口”的标题更改为“易语言”。

#### 2) “边框”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
本属性用于指定窗口边框的类型。边框类型有：0（无边框）；1（普通可调边框）；  
2（普通固定边框）；3（窄标题可调边框）；4（窄标题固定边框）；5（镜框式可调边  
框）；6（镜框式固定边框）。

例如：

```
_启动窗口.边框 = 5
```

程序运行后会将“\_启动窗口”的边框样式改变为镜框式可调边框。

#### 3) “底色”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
本属性用于指定窗口背景的颜色。

例如：

```
_启动窗口.底色 = #红色
```





程序运行后会将“\_启动窗口”的背景颜色设置为“红色”常量所对应的颜色值。

#### 4) “底图”属性

**属性类型：**字节集；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
本属性用于指定显示在窗口背景上的图片。

例如：

```
_启动窗口.底图 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
_启动窗口.底图方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“\_启动窗口”的底图；将底图显示方式设为平铺。

#### 5) “底图方式”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
如果设定了底图）；本属性用于指定窗口背景上底图的显示方式。显示方式有：0（图片居上）；1（图片平铺）；2（图片居中）；3（缩放图片）。

例如：

```
_启动窗口.底图 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
_启动窗口.底图方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“\_启动窗口”的底图；将底图显示方式设为平铺。

#### 6) “背景音乐”属性

**属性类型：**字节集；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当窗口载入时自动开始播放的音乐。被播放的音乐格式可以是MIDI (\*.mid) 或WAV (\*.wav)。

例如：

```
_启动窗口.背景音乐 = 读入文件 (“C:\WINDOWS\Media\flourish.mid”)  
_启动窗口.播放次数 = 0
```

程序运行后会读入“C:\WINDOWS\Media\flourish.mid”（假设文件存在）作为“\_启动窗口”的背景音乐；将背景音乐播放方式设为循环播放。

#### 7) “播放次数”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改  
如果设定背景音乐，本属性用于指定背景音乐的播放次数。本属性可以是以下值之一：0（循环播放）；1（仅播放一次）；2（不播放）。

例如：

```
_启动窗口.背景音乐 = 读入文件 (“C:\WINDOWS\Media\flourish.mid”)  
_启动窗口.播放次数 = 0
```





程序运行后会读入“C:\WINDOWS\Media\flourish.mid”（假设文件存在）作为“\_启动窗口”的背景音乐；将背景音乐播放方式设为循环播放。

#### 8) “控制按钮”属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于设置是否显示窗口上的所有控制按钮及窗口控制菜单。“真”为显示，“假”为不显示。需要注意的是，如果窗口没有标题栏，将看不到控制按钮。

例如：

```
_启动窗口.控制按钮 = 真
_启动窗口.最大化按钮 = 真
_启动窗口.最小化按钮 = 真
```

程序执行后设置“\_启动窗口”的控制按钮为显示；显示最大化控制按钮；显示最小化控制按钮。

#### 9) “最大化按钮”属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

当控制按钮属性为“真”时，本属性用于设置是否显示窗口右上角的最大化或还原按钮。“真”为显示，“假”为不显示或不可用。

例如：

```
_启动窗口.控制按钮 = 真
_启动窗口.最大化按钮 = 真
_启动窗口.最小化按钮 = 真
```

程序执行后设置“\_启动窗口”的控制按钮为显示；显示最大化控制按钮；显示最小化控制按钮。

#### 10) “最小化按钮”属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

当控制按钮属性为“真”时，本属性用于设置是否显示窗口右上角的最小化按钮。“真”为显示，“假”为不显示。

例如：

```
_启动窗口.控制按钮 = 真
_启动窗口.最大化按钮 = 真
_启动窗口.最小化按钮 = 真
```

程序执行后设置“\_启动窗口”的控制按钮为显示；显示最大化控制按钮；显示最小化控制按钮。

#### 11) “位置”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改





本属性用于设置窗口的位置及状态。如果欲设置为“最大化”，“边框”属性必须不为固定类边框。本属性可以是以下值之一：0（通常）；1（居中）；2（最小化）；3（最大化）。

例如：

```
_启动窗口.位置 = 3
```

程序执行后会将“\_启动窗口”最大化。

#### 12) “是否可移动”属性

**属性类型：**逻辑型，**有效范围：**设计时，编程时，**编程时权限：**使用、读取、更改  
本属性用于设置窗口位置能否被移动。“真”为可移动，“假”为不可移动。

例如：

```
_启动窗口.可否移动 = 假
```

程序执行后会禁止“\_启动窗口”移动。

#### 13) “图标”属性

**属性类型：**字节集型，**有效范围：**设计时，编程时，**编程时权限：**使用、读取、更改  
本属性用于设置窗口左上角所显示的图标。支持的格式为ICON (\*.ico)。

例如：

```
_启动窗口.图标 = 读入文件 (“c:\Setup.ico”)
```

程序执行后会读取“c:\Setup.ico”（假设文件存在）作为“\_启动窗口”的图标。

#### 14) “回车下移焦点”属性

**属性类型：**逻辑型，**有效范围：**设计时，编程时，**编程时权限：**使用、读取、更改

本属性如果设置为“真”，运行时当用户在本窗口上按下回车键后，将自动将当前焦点转移到下一个组件。本属性如果设置为“假”，当用户在本窗口上按下回车键后，将等同于按下具有“默认”属性的按钮。

例如：

```
_启动窗口.回车下移焦点 = 真
```

程序执行后会起用按回车移动焦点功能。

#### 15) “Esc键关闭”属性

**属性类型：**逻辑型，**有效范围：**设计时，编程时，**编程时权限：**使用、读取、更改  
本属性用于设置运行时用户是否可以用 Esc 键关闭本窗口。

例如：

```
_启动窗口.Esc键关闭 = 假
```

程序执行后取消按ESC键关闭窗口功能。







## 16) “单击F1键打开帮助”属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于设置运行时用户是否可单击 F1 键打开有关本窗口的帮助信息。为“真”表示打开该功能，为“假”表示关闭该功能。欲打开的帮助文件在“帮助文件名”属性中设置。

例如：

```
_启动窗口.F1键打开帮助 = 真  
_启动窗口.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
_启动窗口.帮助标志值 = 123
```

程序执行后起用在“\_启动窗口”中按F1键打开帮助文件功能；指定所打开的帮助文件名为“C:\Windows\Help\calc.hlp”（假设文件存在，此为Windows计算器帮助文档）；指定显示的帮助标志值为“123”。

## 17) “帮助文件名”属性

**属性类型：**文本型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

如果“F1键打开帮助”属性为“真”时，本属性用于指定欲打开帮助文件的名称，如果没有指定帮助文件，将使用默认帮助文件名称，即本易程序的文件名称（不含其后缀）+“.hlp”后缀。

例如：

```
_启动窗口.F1键打开帮助 = 真  
_启动窗口.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
_启动窗口.帮助标志值 = 123
```

程序执行后起用在“\_启动窗口”中按F1键打开帮助文件功能；指定所打开的帮助文件名为“C:\WINDOWS\Help\calc.hlp”（假设文件存在，此为Windows计算器帮助文档）；指定显示的帮助标志值为“123”。

## 18) “帮助标志值”属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于在帮助文件中指定相应的帮助标志数值。本属性如果为零，则显示帮助主题。

例如：

```
_启动窗口.F1键打开帮助 = 真  
_启动窗口.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
_启动窗口.帮助标志值 = 123
```

程序执行后起用在“\_启动窗口”中按F1键打开帮助文件功能；指定所打开的帮助文件名为“C:\WINDOWS\Help\calc.hlp”（假设文件存在，此为Windows计算器帮助文档）；指定显示的帮助标志值为“123”。



## 19) “在任务条中显示” 属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于设置本窗口是否出现在 Windows 系统的任务条中。为“真”代表显示，“假”代表不显示。

**注解：**当窗口是使用“载入”命令且使用对话框方式加载，无论本属性为何值都不会在任务条中显示。

例如：

`_启动窗口.在任务条中显示 = 假`

程序执行后会取消“\_启动窗口”在任务栏中的显示。

## 20) “随意移动” 属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

如果“可否移动”为“真”，本属性用于设置当用户在窗口上任何位置按下鼠标左键后是否进入窗口整体移动模式。为“真”开启该功能，为“假”关闭该功能。注意如果本属性设置为“真”，则在本窗口及其中所有不能接收输入焦点的子组件上均不能再接收到“鼠标左键被按下”和“鼠标左键被放开”事件。

例如：

`_启动窗口.随意移动 = 真`

程序执行后开启“\_启动窗口”的随意移动功能。

## 21) “外形” 属性

**属性类型：**整数型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于设置窗口的外形。窗口外型可以是以下数值之一：0（矩形）；1（椭圆）；2（圆角矩形）；3（环）；4（正三角）；5（倒三角）；6（左三角）；7（右三角）；8（平行四边形）；9（五边形）；10（六边形）；11（梯形）；12（菱形）；13（五角星）；14（十字星）；15（闪电形）；16（爆炸形1）；17（爆炸形2）；18（燕尾）；19（折角矩形）；20（左箭头）；21（右箭头）；22（上箭头）；23（下箭头）；24（左右箭头）；25（上下箭头）；26（十字箭头）；27（丁字箭头）；28（燕尾箭头）；29（五边形箭头）。

例如：

`_启动窗口.外形 = 16`

程序执行后会将“\_启动窗口”的外形设置为“爆炸形1”。

## 22) “总在最前” 属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于设置本窗口永远位于其他窗口的前面。为“真”代表开启该功能，为“假”代表关闭该功能。





例如：

**\_启动窗口. 总在最前 = 真**

程序执行后开启“\_启动窗口”总在最前的功能。

23) “保持标题条激活”属性

**属性类型：**逻辑型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

如果希望本窗口在非激活状态下其标题条仍然与激活状态时相同，请设置本属性为真。

例如：

**\_启动窗口. 保持标题条激活 = 真**

程序执行后设置“\_启动窗口”始终保持激活状态。

24) “窗口类名”属性

**属性类型：**文本型；**有效范围：**设计时，编程时；**编程时权限：**使用、读取、更改

本属性用于指定本窗口在创建时所使用的窗口类名。本属性仅设计时设置有效，如果在编程时改变本属性，只会改变本属性的值，但窗口类名并不随之改变。

例如：

▶ **输出调试文本 (\_启动窗口. 窗口类名)**

程序运行后在输出窗口中显示“\_启动窗口”的窗口类名。

#### 4.1.4 共有事件

事件类似于一个子程序，当程序运行时接受到某些信息时系统会自动调用这些子程序以便程序进行处理这些信息。

在易语言中，所有的可见组件（运行时）都拥有一些相同的事件，即共有事件。下面将逐一介绍。

##### 4.1.4.1 不需要停留焦点的共有事件

1) “鼠标左键被按下”事件

事件原型为：

**返回值类型：**逻辑型

**参数一：**参数名：横向位置 **类型：**整数型

**参数二：**参数名：纵向位置 **类型：**整数型

**参数三：**参数名：功能键状态 **类型：**整数型

当鼠标在事件对应的对象上按下鼠标左键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。





参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下的状态。

可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_鼠标左键被按下	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

--- 如果真（位与（功能键状态，#Alt键状态）≠ 0）  
» 输出调试文本（横向位置）  
» 输出调试文本（纵向位置）  
返回（假）

程序运行后当在“\_启动窗口”上按下鼠标左键，程序会判断是否这时同时按下了Alt键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量；处理后指定了不将消息传递给系统默认处理函数。

## 2) “鼠标左键被放开”事件

事件原型为：

返回值类型：逻辑型
参数一：参数名：横向位置 类型：整数型
参数二：参数名：纵向位置 类型：整数型
参数三：参数名：功能键状态 类型：整数型

当鼠标在事件对应的对象上放开鼠标左键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或



其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_鼠标左键被放开	逻辑型				
参数名	类 型	参 考	可 空	数 组	备 注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

--- 如果真（位与（功能键状态，#Ctrl键状态）≠ 0）  
» 输出调试文本（横向位置）  
» 输出调试文本（纵向位置）  
返回（假）

程序运行后当在“\_启动窗口”上放开鼠标左键，程序会判断是否这时同时按下了Ctrl键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量。处理后指定了不将消息传递给系统默认处理函数。

### 3) “被双击”事件

事件原型为：

返回值类型：逻辑型

参数一：参数名：横向位置 类型：整数型

参数二：参数名：纵向位置 类型：整数型

参数三：参数名：功能键状态 类型：整数型

当鼠标在事件对应的对象上双击鼠标左键时触发本事件。对双击的判定使用当前所在操作系统中的设置。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。



例如：

子程序名	返回值类型	公开	备注		
__启动窗口_被双击	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

--- 如果真 (位与 (功能键状态, #Shift键状态) ≠ 0)

» 输出调试文本 (横向位置)

» 输出调试文本 (纵向位置)

» 返回 (假)

程序运行后当在“\_启动窗口”上双击鼠标左键，程序会判断是否这时同时按下了Shift键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量。处理后指定了不将消息传递给系统默认处理函数。

4) “鼠标右键被按下”事件

事件原型为：

返回值类型：逻辑型
参数一：参数名：横向位置 类型：整数型
参数二：参数名：纵向位置 类型：整数型
参数三：参数名：功能键状态 类型：整数型

当鼠标在事件对应的对象上按下鼠标右键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_鼠标右键被按下	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				





```

    ... 如果真 (位与 (功能键状态, #Ctrl键状态) ≠ 0)
    >> 输出调试文本 (横向位置)
    >> 输出调试文本 (纵向位置)
    ↓
    返回 (假)

```

程序运行后当在“\_启动窗口”上按下鼠标右键，程序会判断是否这时同时按下了Ctrl键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量。处理后指定了不将消息传递给系统默认处理函数。

#### 5) “鼠标右键被放开”事件

事件原型为：

返回值类型：逻辑型

参数一：参数名：横向位置 类型：整数型

参数二：参数名：纵向位置 类型：整数型

参数三：参数名：功能键状态 类型：整数型

当鼠标在事件对应的对象上放开鼠标右键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_鼠标右键被放开	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

```

    ... 如果真 (位与 (功能键状态, #Alt键状态) ≠ 0)
    >> 输出调试文本 (横向位置)
    >> 输出调试文本 (纵向位置)
    ↓
    返回 (假)

```





程序运行后当在“\_启动窗口”上放开鼠标右键，程序会判断是否这时同时按下了Alt键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量。处理后指定了不将消息传递给系统默认处理函数。

6) “鼠标位置被移动”事件

事件原型为：

- 返回值类型：逻辑型
- 参数一：参数名：横向位置 类型：整数型
- 参数二：参数名：纵向位置 类型：整数型
- 参数三：参数名：功能键状态 类型：整数型

当鼠标在事件对应的对象上位置被改变时触发本事件。如果鼠标一直在移动，触发本事件的间隔为鼠标采样率。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标指针相对于对应对象的左上角横向（x）偏移量（相对位置）。

参数<2>保存了事件触发时鼠标指针相对于对应对象的左上角纵向（y）偏移量（相对位置）。

参数<3>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和： 1（#Ctrl键状态）； 2（#Shift键状态）； 4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_鼠标位置被移动	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

--- 如果真 (位与 (功能键状态, #Alt键状态) ≠ 0)

» 输出调试文本 (横向位置)

» 输出调试文本 (纵向位置)

↓ 返回 (假)

程序运行后当在“\_启动窗口”上移动鼠标时，程序会判断是否这时同时按下了Alt键，如果是则在输出面板中显示鼠标相对于“\_启动窗口”的左上角位置的偏移量。处理后，指定了不将消息传递给系统默认处理函数。



### 4.1.4.2 需要停留焦点的共有事件

需要焦点停留在该对象时才会触发它的事件。

#### 1) “滚轮被滚动”事件

事件原型为：

返回值类型：逻辑型

参数一：参数名：滚动距离 类型：整数型

参数二：参数名：功能键状态 类型：整数型

事件对应的对象拥有焦点且当鼠标在事件对应的对象上滚动鼠标滚轮时触发本事件。可使用“对象.获取焦点（）”命令来强制使某对象拥有焦点。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时鼠标滚轮相对移动距离，大于零表示向前滚动，小于零表示向后滚动。

参数<2>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_滚轮被滚动	逻辑型				
参数名	类 型	参考	可空	数组	备 注
滚动距离	整数型				
功能键状态	整数型				

```

-- 如果真 (位与 (功能键状态, #Alt键状态) ≠ 0)
  输出调试文本 (滚动距离)
  返回 (假)
  
```

程序运行后当在“\_启动窗口”上滚动鼠标时，程序会判断是否同时按下了Alt键，如果是则在输出面板中显示鼠标滚轮滚动的距离。处理后指定了不将消息传递给系统默认处理函数。

#### 2) “获得焦点”事件

事件原型为：

无返回值；无参数

事件对应的对象拥有焦点时触发本事件。可使用“对象.获取焦点（）”命令来强制使某对象拥有焦点。



例如：

子程序名	返回值类型	公开	备注
__启动窗口_获得焦点			

» 输出调试文本（“获得焦点”）

程序运行后当“\_\_启动窗口”获得焦点时，在输出面板中显示“获得焦点”。

3) “失去焦点”事件

事件原型为：

无返回值；无参数

事件对应的对象拥有的焦点失去时触发本事件。

例如：

子程序名	返回值类型	公开	备注
__启动窗口_失去焦点			

» 输出调试文本（“失去焦点”）

程序运行后当“\_\_启动窗口”失去焦点时，在输出面板中显示“失去焦点”。

4) “按下某键”事件

事件原型为：

返回值类型：逻辑型

参数一：参数名：键代码 类型：整数型

参数二：参数名：功能键状态 类型：整数型

事件对应的对象拥有焦点且当按下键盘按键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

**注解：**如果返回“假”，会影响到“字符输入”事件的触发。

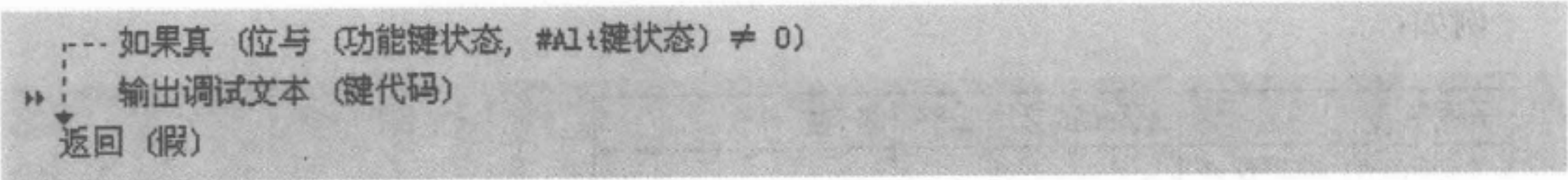
参数<1>保存了事件触发时按下的键对应的键代码。

参数<2>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_按下某键	逻辑型				
参数名	类型	参考	可空	数组	备注
键代码	整数型				
功能键状态	整数型				





程序运行后当在“\_启动窗口”拥有焦点并按下某键时，程序会判断是否这时同时按下了Alt键，如果是则在输出面板中显示按下的键对应的键代码，注意此时键代码并不包含Alt键。处理后指定了不将消息传递给系统默认处理函数。

5) “放开某键”事件

事件原型为：

返回值类型：逻辑型  
参数一：参数名：键代码 类型：整数型  
参数二：参数名：功能键状态 类型：整数型

事件对应的对象拥有焦点且当放开键盘按键时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“真”代表传递，返回“假”代表不传递，默认为返回“真”。

参数<1>保存了事件触发时按下的键对应的键代码。

参数<2>保存了事件触发时各功能键是否被同时按下的状态。可以为以下常量值或其和：1（#Ctrl键状态）；2（#Shift键状态）；4（#Alt键状态）。要判断是否同时按下了某个功能键，在此参数的值与想要得知的功能键的状态常量值之间使用“位与”命令进行比较。如果所得的结果不为零，则表示同时按下了此功能键。

例如：

子程序名	返回值类型	公开	备注		
__启动窗口_放开某键	逻辑型				
参数名	类型	参考	可空	数组	备注
键代码	整数型				
功能键状态	整数型				

```
graph TD
    A[如果真 (位与 (功能键状态, #Alt键状态) ≠ 0)] -- 真 --> B[输出调试文本 (键代码)]
    A -- 假 --> C[返回 (假)]
```

程序运行后当在“\_启动窗口”拥有焦点并放开某键时，程序会判断是否这时同时按下了Alt键，如果是则在输出面板中显示按下的键对应的键代码，注意此时键代码并不包含Alt键。处理后指定了不将消息传递给系统默认处理函数。

6) “字符输入”事件

事件原型为：

返回值类型：整数型  
参数一：参数名：字符代码 类型：整数型



事件对应的对象拥有焦点且当完成一个字符输入时触发本事件。

本事件的返回值用于控制事件执行完毕后是否向系统中默认的事件处理函数传递本事件。返回“0”代表通知默认函数无字符输入，默认为返回参数一的值。

**注解：**如果返回0，将对“编辑框”等组件的输入造成影响。

参数<1>保存了事件触发时按下的字符对应的字符代码（ASCII码）。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_字符输入	整数型				
参数名	类 型	参 考	可 空	数 组	备 注
字符代码	整数型				

» 输出调试文本（字符（字符代码））  
返回（0）

程序运行后当在“\_启动窗口”拥有焦点并完成一个字符输入时，程序会将字符代码转换为文字显示在输出面板中。

4.1.5 独有事件

1) “创建完毕”事件

事件原型为：

无返回值  
无参数

当窗口及其中的所有组件均被创建后在显示之前产生此事件，用户可以在响应此事件期间做一些初始化工作。本事件为该窗口触发的第一个事件。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_创建完毕			

» 输出调试文本（“创建完毕事件被触发”）

程序运行后，当“\_启动窗口”创建完毕时，在输出窗口中显示“创建完毕事件被触发”。

2) “可否被关闭”事件

事件原型为：

返回值类型：逻辑型  
无参数

在窗口被关闭之前产生此事件，用于询问窗口可否被关闭。如果返回“假”则不允许窗口被关闭，返回“真”或不返回值允许关闭。



例如：

子程序名	返回值类型	公开	备 注
__启动窗口_可否被关闭	逻辑型		

» 输出调试文本（“可否被关闭 被触发”）  
返回（假）

程序运行后，当试图关闭“\_启动窗口”时，将在输出窗口中显示“可否被关闭 被触发”，并返回“假”不允许窗口关闭。

### 3) “将被销毁”事件

事件原型为：

无返回值

无参数

当窗口将被销毁前产生此事件。窗口被销毁后所有对此窗口及其内部组件的修改都将被作废。本事件与“可否被关闭”事件不同之处在于，本事件一旦触发意味着窗口即将不可用，而“可否被关闭”事件可以通过返回“假”来继续使用该窗口。本事件一般用于结束一个功能界面时进行相关的资源释放操作。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_将被销毁			

» 输出调试文本（“将被销毁 被触发”）

程序运行后，当“\_启动窗口”将被销毁时在输出窗口中显示“将被销毁 被触发”

### 4) “位置被改变”事件

事件原型为：

无返回值

无参数

当窗口的位置被改变时产生此事件。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_位置被改变			

» 输出调试文本（“位置被改变 被触发”）

程序运行后，当“\_启动窗口”的位置被改变时，在输出窗口中显示“位置被改变 被触发”。

### 5) “尺寸被改变”事件

事件原型为：



无返回值

无参数

当窗口的尺寸被改变后产生此事件。

例如：

子程序名	返回值类型	公开	备注
_启动窗口_尺寸被改变			

» 输出调试文本（“尺寸被改变 被触发”）

程序执行后当“\_启动窗口”的大小被改变时在输出窗口总显示“尺寸被改变 被触发”。

### 6) “被激活”事件

事件原型为：

无返回值

无参数

当窗口转换为激活状态时产生此事件。

例如：

子程序名	返回值类型	公开	备注
_启动窗口_被激活			

» 输出调试文本（“被激活 被触发”）

程序运行后当“\_启动窗口”被激活时在输出窗口中显示“被激活 被触发”。

### 7) “被取消激活”事件

事件原型为：

无返回值

无参数

当窗口转换为非激活状态时产生此事件。

例如：

子程序名	返回值类型	公开	备注
_启动窗口_被取消激活			

» 输出调试文本（“被取消激活 被触发”）

程序运行后当“\_启动窗口”失去激活状态时在输出窗口中显示“被取消激活 被触发”。

### 8) “空闲”事件

事件原型为：

返回值类型：逻辑型

参数一：参数名：已空闲时间类型：整数型



当系统正处于空闲状态时产生此事件，用户可以在响应此事件时做一些后台处理工作。如果事件处理子程序返回“假”或者不返回值，在此次空闲期间系统将不再产生空闲事件（可以降低 CPU 占用率）。如果返回“真”，系统将产生空闲事件。

参数<1>指示自系统本次进入空闲状态后到现在所经过的空闲时间，单位为毫秒。用户程序在空闲状态下所进行的大工作量操作最好等到此值较大时进行。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_空闲	逻辑型				
参数名	类 型	参考	可空	数组	备 注
已空闲时间	整数型				

» 输出调试文本（“空闲 被触发，空闲时间为：” + 到文本（已空闲时间））  
返回（真）

程序执行后，当无用户操作时将在输出窗口中显示已空闲的时间。

9) “首次激活” 事件

事件原型为：

无返回值

无参数

当窗口创建完毕后被首次激活显示时产生此事件，在此事件及此事件后用户可以开始在窗口内的画板组件上写出内容。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_首次激活			

» 输出调试文本（“首次激活 被触发”）

程序运行后，当“\_启动窗口”被首次激活时在输出窗口中显示一段文字。

10) “被显示” 事件

事件原型为：

无返回值

无参数

当窗口被显示时产生此事件。

例如：

子程序名	返回值类型	公开	备 注
__启动窗口_被显示			

» 输出调试文本（“被显示 被触发”）



程序运行后，当“\_启动窗口”被显示时在输出窗口中显示一段文字。

11) “被隐藏”事件

事件原型为：

无返回值  
无参数

当窗口被隐藏时（例如“可视”属性为“假”）产生此事件。

例如：

子程序名	返回值类型	公开	备注
__启动窗口_被隐藏			

» 输出调试文本（“被隐藏 被触发”）

程序运行后，当“\_启动窗口”被隐藏时在输出窗口中显示一段文字。

12) “托盘事件”事件

事件原型为：

返回值类型：逻辑型  
参数一：参数名：操作类型类型：整数型

当程序使用“置托盘图标”设置了程序托盘图标后，当用户用鼠标操作本程序托盘图标后即产生本事件。

参数<1>指示用户操作类型，可能为的值为：1（#单击左键）；2（#双击）；3（#单击右键）。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_托盘事件					
参数名	类 型	参 考	可空	数组	备 注
操作类型	整数型				

» 判断 (操作类型 = 1) ' 左键单击  
» 输出调试文本（“左键单击”）  
» 判断 (操作类型 = 2) ' 左键双击  
» 输出调试文本（“左键双击”）  
» 判断 (操作类型 = 3) ' 右键单击  
» 输出调试文本（“右键单击”）

程序运行后，当用户操作本程序的托盘图标时，在输出窗口中显示用户所做的操作。  
本例程源码见随书光盘中“\图书例程\第四章\托盘事件.e”

4.1.6 共有命令

正如前文所说，窗口与窗口组件都是同源的产物，所以窗口所拥有的命令所有窗口组



件都可以使用。下面将介绍窗口组件的共有命令，也是窗口所拥有的所有命令。

**注解：**调用窗口和窗口组件命令的方式为：

<返回值> 对象.命令名([参数])

若是在窗口对应的程序集中欲对此窗口进行调用时，可不加对象名直接使用此命令：

<返回值> 命令名([参数])

#### 1) “取窗口句柄”命令

命令原型为：

<整数型> 对象.取窗口句柄 ()

本命令用于取出本窗口或窗口组件的窗口句柄（即HWND）。窗口句柄一般用于配合Windows API时使用。

**注解：**HWND 是 Handle of Window的缩写，就是窗口句柄的意思。句柄就是唯一表示某种事物的一个号码，就像身份证一样，Windows里当前打开的所有窗口都有自己的ID，那个ID就是句柄。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = \_启动窗口.取窗口句柄 ()

→ 输出调试文本 (变量1)

程序运行后会取得“\_启动窗口”的窗口句柄并存放在“变量1”中；在输出面板中显示“变量1”中的内容。

#### 2) “销毁”命令

命令原型为：

<无返回值> 对象.销毁 ([逻辑型 立即销毁])

本命令用于销毁本窗口或指定的窗口组件。窗口被销毁后，所有在载入本窗口之后对本窗口及其内窗口组件所进行的修改或设置都将被抛弃，窗口被销毁后如想再次使用必须重新装载。

**注解：**当窗口调用本命令进行销毁时，窗口并不会被立即销毁，而会被推迟到其他所有现存窗口事件处理完毕后才会被真正销毁。如果被销毁的是拥有子窗口（或窗口组件）的父窗口，则其所有子窗口（或窗口组件）将被一同销毁。

参数<1>指定了是否强制进行销毁。通常情况下，调用销毁命令后为了考虑到事件处理子程序的后续安全操作，窗口真正的销毁工作会被延迟到所有事件处理子程序执行完毕后再进行，但有时由于某种特殊需要，可能希望窗口能够立即被销毁，设置本参数为真即可。

**注解：**核心库4.6版本以前（不包括4.6）由于不支持本参数，执行本命令始终会立即销毁；本参数仅在销毁窗口时有效，销毁窗口组件时始终都采取立即销毁方式。如果被省略，则参数默认值为假。





例如：

\_启动窗口.销毁 ()

程序执行后会销毁“\_启动窗口”。

**注解：** 当在系统配置中设置使用“\_启动窗口”作为“Windows程序启动方式”时，销毁“\_启动窗口”意味着结束整个程序的运行。

3) “获取焦点”命令

命令原型为：

<无返回值> 对象.获取焦点 ()

本命令用于将焦点移动到窗口组件。如果对窗口使用本命令，窗口将自动把焦点转移到第一个有能力保留焦点的窗口组件上去。调用本命令会触发目标对象的“获得焦点”事件。

例如：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

\_启动窗口.获取焦点 ()

程序运行后将焦点移动到“\_启动窗口”上。

4) “可有焦点”命令

命令原型为：

<逻辑型> 对象.可有焦点 ()

本命令用于判断焦点是否在调用对象上。如果当前对象具有焦点，则返回“真”，否则返回“假”。

例如：

变量名	类型	静态	数组	备注
变量1	逻辑型			

变量1 = \_启动窗口.可有焦点 ()

输出调试文本 (变量1)

程序执行后会判断“\_启动窗口”当前是否获得焦点，并将判断结果保存在“变量1”中；在输出面板中显示“变量1”的内容。

5) “取用户区宽度”命令

命令原型为：

<整数型> 对象.取用户区宽度 ()

本命令用于获取窗口或窗口组件用户区域的宽度，单位为像素点。对窗口而言；用户区域为窗口区域减去标题栏、菜单栏、边框后的区域；对窗口组件而言，用户区域等同于其窗口组件区域。





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = \_启动窗口.取用户区宽度 0

» 输出调试文本 (变量1)

程序运行后会取得“\_启动窗口”的用户区宽度并保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 6) “取用户区高度”命令

命令原型为：

<整数型> 对象.取用户区高度 ()

本命令用于获取窗口或窗口组件用户区域的高度，单位为像素点。对窗口而言，用户区域为窗口区域减去标题栏、菜单栏、边框后的区域；对窗口组件而言，用户区域等同于其窗口组件区域。

例如：

变量名	类 型	静态	数组	备 注
变量1	整数型			

变量1 = \_启动窗口.取用户区高度 0

» 输出调试文本 (变量1)

程序运行后会取得“\_启动窗口”的用户区高度并保存在“变量1”中；在输出面板中显示“变量1”的内容。

#### 7) “禁止重画”命令

命令原型为：

<无返回值> 对象.禁止重画 ()

本命令用于禁止易语言系统对窗口或窗口组件自动重画，显示内容的改变会在下次系统重画时显现。本命令一般用于以避免当频繁进行操作时引起闪烁，先让其禁止自动重画，待最终需要显示时恢复重画。调用本命令后如需要恢复系统自动重画需要调用“允许重画”命令。但本命令只能控制在对对象显示内容改变后不发送重画指令（消息），并不能阻止系统内部对重画的处理。所以，如果接收来自Windows系统的重画指令（消息）时（如遮盖、移动等），还是会处理相关的重画工作，如图4-7所示。

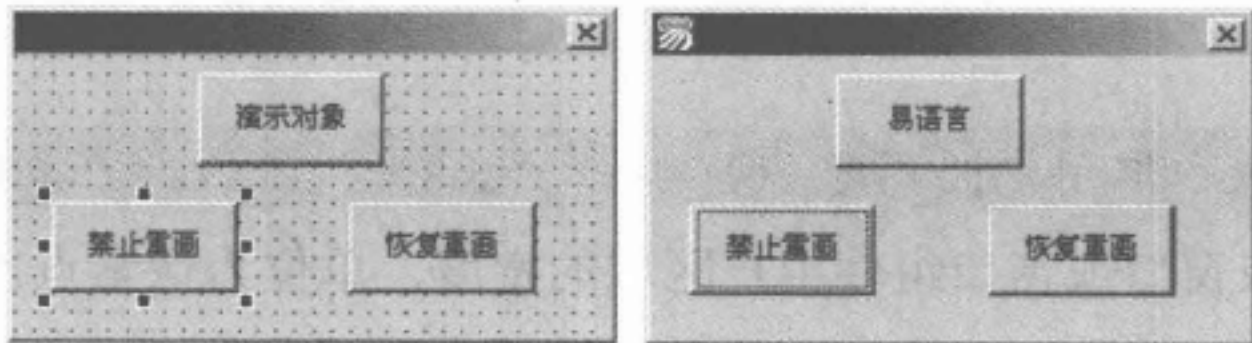


图4-7 重画演示

在“禁止重画”按钮和“恢复重画”按钮的单击事件里输入如下代码：





子程序名	返回值类型	公开	备注
按钮_禁止重画_被单击			

按钮\_演示对象.禁止重画 ()

按钮\_演示对象.标题 = “易语言”

子程序名	返回值类型	公开	备注
按钮_恢复重画_被单击			

按钮\_演示对象.允许重画 ()

程序运行后单击“禁止重画”按钮，禁止“演示对象”按钮进行自动重画；改变“演示对象”的标题为“易语言”；

这时标题的改变效果不会马上显现，但重画任务会进入后台队列，等待系统通知重画。

单击“恢复重画”按钮允许系统进行自动重画；“演示对象”的标题改变结果将会被显现。

本例程源码见随书光盘中“\图书例程\第四章\重画演示\重画示例1.e”。

#### 8) “允许重画”命令

命令原型为：

<无返回值> 对象.允许重画 ()

本命令用于重新允许被禁止的窗口或窗口组件自动重画。本命令被调用后会自动向该对象增加一次重画任务，如图4-8所示。

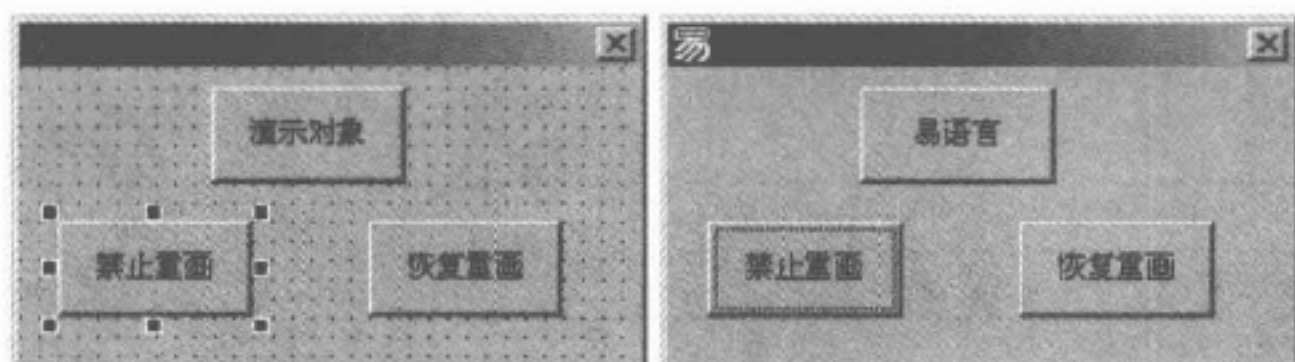


图4-8 重画演示

例如：

子程序名	返回值类型	公开	备注
按钮_禁止重画_被单击			

按钮\_演示对象.禁止重画 ()

按钮\_演示对象.标题 = “易语言”

子程序名	返回值类型	公开	备注
按钮_恢复重画_被单击			

按钮\_演示对象.允许重画 ()

程序运行后单击“禁止重画”按钮，禁止“演示对象”按钮进行自动重画；改变“演示对象”的标题为“易语言”。

这时标题的改变效果不会马上显现，但重画任务会进入后台队列，等待系统通知重画。



单击“恢复重画”按钮允许系统进行自动重画；“演示对象”的标题改变结果将会被显现。

本例程源码见随书光盘中“\图书例程\第四章\重画演示\重画示例1.e”。

### 9) “重画”命令

命令原型为：

<无返回值> 对象. 重画 ()

本命令用于通知 Windows 系统本窗口或窗口组件上的显示内容需要在以后被全部重画，即向系统内部增加一个重画任务。本命令增加的重画任务不会被马上执行，并可以使用“取消重画”命令取消所有未执行的重画任务，如图4-9所示。

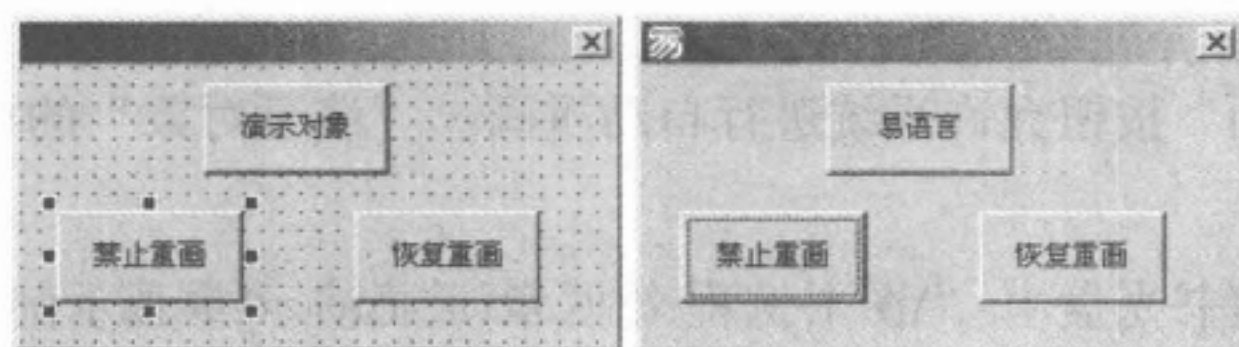


图4-9 重画演示

例如：

子程序名	返回值类型	公开	备注
_按钮_恢复重画_被单击			

按钮\_演示对象. 允许重画 ()

按钮\_演示对象. 重画 ()

延时 (5000) ' 用于模拟耗时代码

子程序名	返回值类型	公开	备注
_按钮_禁止重画_被单击			

按钮\_演示对象. 禁止重画 ()

按钮\_演示对象. 标题 = “易语言”

程序运行后单击“禁止重画”按钮，禁止“演示对象”按钮自动重画；改变“演示对象”的标题为“易语言”；

单击“恢复重画”按钮允许系统进行自动重画；通知系统重画“演示对象”；让程序在此停留5秒，用于模拟耗时代码，可以看到这时“演示对象”并没有被更新；5秒后“演示对象”的标题改变结果被显现。

本例程源码见随书光盘中“\图书例程\第四章\重画演示\重画示例2.e”

### 10) “部分重画”命令

命令原型为：

<无返回值> 对象. 部分重画 (整数型 欲重画区域的左边, 整数型 欲重画区域的顶边, 整数型 欲重画区域的宽度, 整数型 欲重画区域的高度)

本命令用于通知 Windows 系统本窗口或窗口组件上的显示内容需要在以后被部分重





画。本命令增加的重画任务不会被马上执行，并可以使用“取消重画”命令取消所有未执行的重画任务，如图4-10所示。

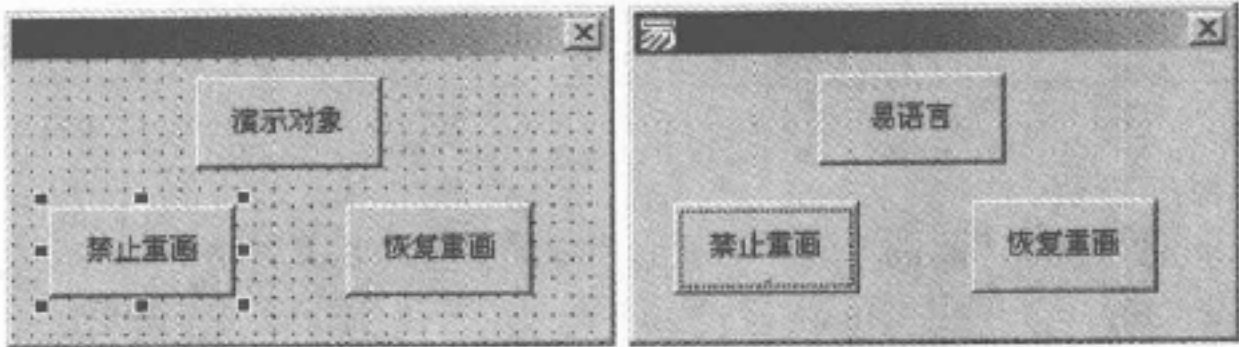


图4-10 重画演示

例如：

子程序名	返回值类型	公开	备注
按钮_恢复重画_被单击			

按钮\_演示对象.允许重画 ()  
按钮\_演示对象.取消重画 ()  
' 因“允许重画”会自动调用一次“重画”，所以使用“取消重画”将其取消  
按钮\_演示对象.部分重画 (0, 0, 按钮\_演示对象.宽度 ÷ 2, 按钮\_演示对象.高度)  
延时 (5000) ' 用于模拟耗时代码

子程序名	返回值类型	公开	备注
按钮_禁止重画_被单击			

按钮\_演示对象.禁止重画 ()  
按钮\_演示对象.标题 = “易语言”

程序运行后单击“禁止重画”按钮，禁止“演示对象”自动重画；改变“演示对象”的标题为“易语言”。

单击“恢复重画”按钮允许系统进行自动重画；取消由“允许重画”命令自动添加的重画任务；通知系统重画“演示对象”左半部分；让程序在此停留5秒，用于模拟耗时代码，可以看到这时“演示对象”并没有被更新；5秒后“演示对象”的标题改变结果被显现。

本例程源码见随书光盘中“\图书例程\第四章\重画演示\重画示例3.e”

11) “取消重画”命令

命令原型为：

<无返回值> 对象.取消重画 ()

本命令用于通知 Windows 系统本窗口或窗口组件不再需要被重画（取消将要执行的重画任务），保留现有的全部显示内容，如图4-11所示。

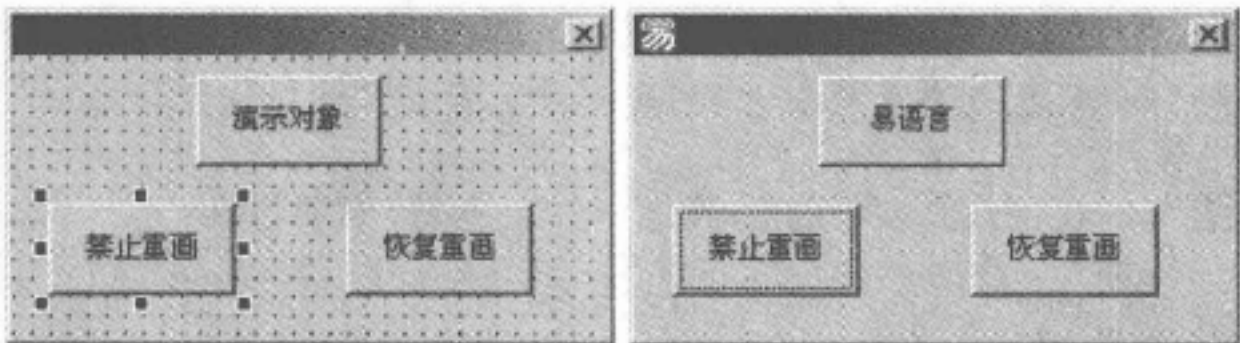


图4-11 重画演示



例如：

子程序名	返回值类型	公开	备注
_按钮_恢复重画_被单击			

按钮\_演示对象.允许重画 0  
 按钮\_演示对象.取消重画 0  
 ' 因“允许重画”会自动调用一次“重画”，所以使用“取消重画”将其取消  
 按钮\_演示对象.部分重画 (0, 0, 按钮\_演示对象.宽度 ÷ 2, 按钮\_演示对象.高度)  
 延时 (5000) ' 用于模拟耗时代码

子程序名	返回值类型	公开	备注
_按钮_禁止重画_被单击			

按钮\_演示对象.禁止重画 0  
 按钮\_演示对象.标题 = “易语言”

程序运行后单击“禁止重画”按钮，禁止“演示对象”自动重画；改变“演示对象”的标题为“易语言”；

单击“恢复重画”按钮允许系统进行自动重画；取消由“允许重画”命令自动添加的重画任务；通知系统重画“演示对象”左半部分；让程序在此停留5秒，用于模拟耗时代码，可以看到这时“演示对象”并没有被更新；5秒后“演示对象”的标题改变结果被显现。

本例程源码见随书光盘中 “\图书例程\第四章\重画演示\重画示例3.e”

## 12) “刷新显示” 命令

命令原型为：

<无返回值> 对象.刷新显示 ()

本命令用于通知Windows系统立即执行对本对象的重画任务，如图4-12所示。

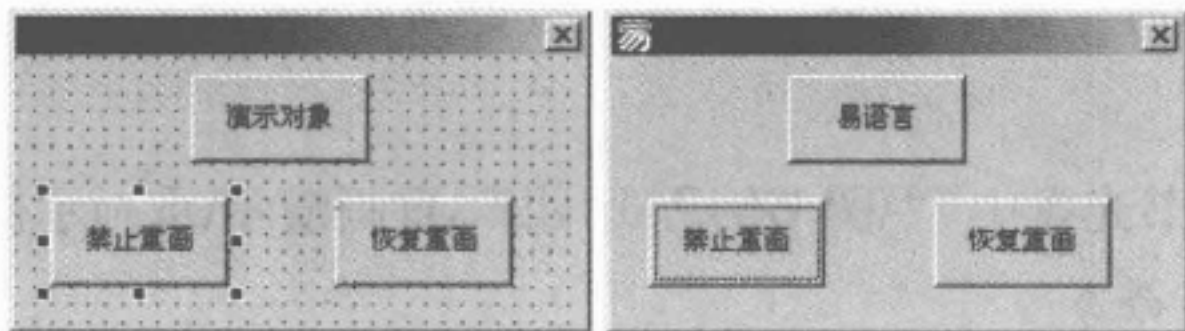


图4-12 重画演示

例如：

子程序名	返回值类型	公开	备注
_按钮_恢复重画_被单击			

按钮\_演示对象.允许重画 0  
 按钮\_演示对象.取消重画 0 ' 因允许重画会自动调用一次“重画”命令，  
 所以使用取消重画将其取消  
 按钮\_演示对象.部分重画 (0, 0, 按钮\_演示对象.宽度 ÷ 2, 按钮\_演示对象.高度)  
 按钮\_演示对象.刷新显示 0  
 延时 (5000) ' 用于模拟耗时代码





子程序名	返回值类型	公开	备注
按钮_禁止重画_被单击			

按钮\_演示对象.禁止重画 ()  
按钮\_演示对象.标题 = “易语言”

程序运行后单击“禁止重画”按钮，禁止“演示对象”自动重画；改变“演示对象”的标题为“易语言”；

单击“恢复重画”按钮允许系统进行自动重画；取消未执行的重画任务；重画按钮\_演示对象的左半部分；通知系统立即执行重画任务，可以看到这时“演示对象”被立即更新；让程序在此停留5秒，用于模拟耗时代码。

本例程源码见随书光盘中“\图书例程\第四章\重画演示\重画示例4.e”；另可以和“\图书例程\第四章\重画演示\重画示例2.e”进行执行效果的对比。

13) “移动”命令

命令原型为：

<无返回值> 对象.移动 ([整数型 左边], [整数型 顶边], [整数型 宽度], [整数型 高度])

本命令用于改变窗口或窗口组件的位置或尺寸。

参数<1>指定了移动后对象新位置左上角相对于其父窗口左上角的横向偏移量(x)。单位为像素点。如被省略则不改变左边原位置。

参数<2>指定了移动后对象新位置左上角相对于其父窗口左上角的纵向偏移量(y)。单位为像素点。如被省略则不改变顶边原位置。

参数<3>指定了对象新的宽度。单位为像素点。如被省略或等于-1，则不改变原宽度。

参数<4>指定了对象新的高度。单位为像素点。如被省略或等于-1，则不改变原高度。

例如：

\_启动窗口.移动(, , 50, 50)

程序运行后会改变“\_启动窗口”的宽度和高度都为50像素。

14) “调整层次”命令

命令原型为：

<无返回值> 对象.调整层次 ([整数型 欲调整到的层次])

本命令用于改变窗口或窗口组件的现行所处层次。

参数<1>指定了调整到的目标层。

本参数可以为以下常量之一：1（#顶层）；2（#底层）；3（#最高层）；4（#次高层）。如果被省略，默认为“#顶层”。



标签层次演示如图4-13所示。

调整层次标签.调整层次(#顶层)

程序执行后单击按钮，标签二会被移动到最顶层（靠近屏幕）。

本例程源码见随书光盘中“\图书例程\第四章\调整层次.e”

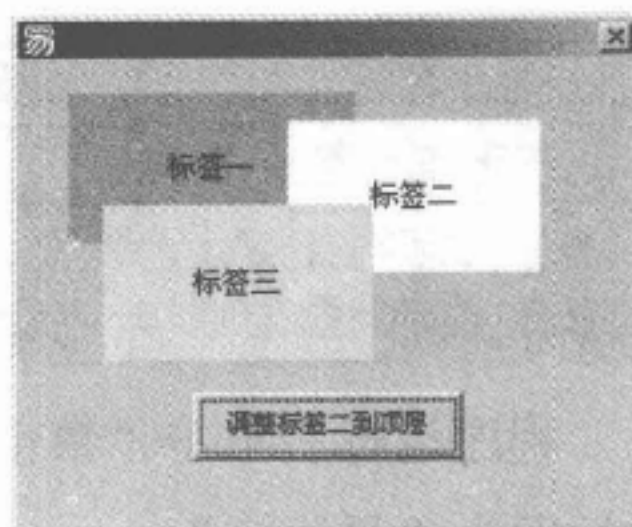


图4-13 调整层次演示

## 15) “弹出菜单”命令

命令原型:

<无返回值> 对象. 弹出菜单 (菜单 欲弹出的菜单, [整数型 水平显示位置], [整数型 垂直显示位置])

本命令用于在窗口上的当前鼠标位置或指定坐标位置显示弹出式菜单，如果调用对象为窗口组件，则自动作用其所在的窗口。

参数<1>指定了欲弹出的菜单。指定菜单内必须包含有子菜单，且所处窗口必须与调用对象窗口一致。

参数<2>指定了菜单弹出时的横向位置(x)。单位为像素点，相对于屏幕左边。如果被省略，将自动使用当前鼠标位置。

参数<3>指定了菜单弹出时的纵向位置(y)。单位为像素点，相对于屏幕顶边。如果被省略，将自动使用当前鼠标位置。

例如:

子程序名	返回值类型	公开	备注		
__启动窗口_鼠标左键被按下	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

弹出菜单 (菜单, , )

程序运行后，在“\_启动窗口”中按下鼠标左键时在当前鼠标位置处弹出“菜单”中的子菜单项。

本例程源码见随书光盘中“\图书例程\第四章\弹出菜单.e”

## 16) “发送信息”命令

命令原型为:

<整数型> 对象. 发送信息 (整数型 信息值, 整数型 参数1, 整数型 参数2)

本命令用于将指定Windows消息发送到窗口或窗口组件，并等待且取回信息反馈值。本命令是对Windows API — SendMessage命令的封装。本命令根据所提供的参数不同，所执行的效果不同。有关Windows消息请查阅相关资料。



**注解：**API (Application Programming Interface) 的中文含义是应用程序编程接口。其实，它是程序员与Windows交互的最基本方式，包含了所有应用程序对构造操作系统的函数的调用。

参数<1>指定了欲向指定对象发送的Windows消息值。

参数<2>指定了该消息的第一个参数值，初始值为“0”。

参数<3>指定了该消息的第二个参数值，初始值为“0”。

例如：

变量名	类型	静态	数组	备注
变量1	整数型			

变量1 = 编辑框1.发送信息 (193, 0, 0)

» 输出调试文本 (变量1)

程序执行后向“编辑框1”发送“193”号消息，作用为获取当前光标所在行的字符数，并将结果保存在“变量1”中；在输出面板中显示“变量1”中的内容，如图4-14所示。

本例程源码见随书光盘中“\图书例程\第四章\发送信息.e”

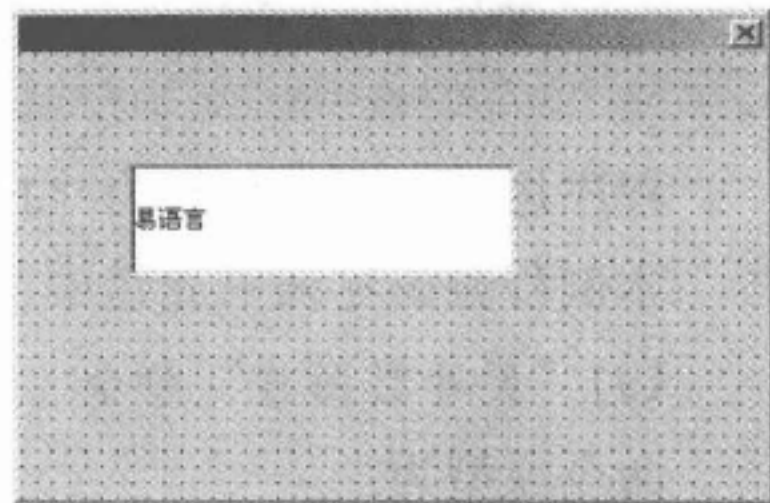


图4-14 例程界面

#### 17) “投递信息”命令

命令原型为：

<无返回值> 对象. 投递信息 (整数型 信息值, 整数型 参数1, 整数型 参数2)

本命令用于将指定Windows消息发送到窗口或窗口组件，但不等待命令返回。本命令是对Windows API — PostMessage命令的封装。本命令根据所提供的参数不同，所执行的效果不同。有关Windows消息请查阅相关资料。

参数<1>指定了欲向指定对象发送的Windows消息值。

参数<2>指定了该消息的第一个参数值，初始值为“0”。

参数<3>指定了该消息的第二个参数值，初始值为“0”。

例如：

子程序名	返回值类型	公开	备 注		
__启动窗口_鼠标左键被按下	逻辑型				
参数名	类 型	参考	可空	数组	备 注
横向位置	整数型				
纵向位置	整数型				
功能键状态	整数型				

\_启动窗口.投递信息 (16, 0, 0) ' 投递销毁窗口的消息

程序运行后在“\_启动窗口”上按下鼠标左键后向“\_启动窗口”发送“16”号消息，作用为销毁窗口。

本例程源码见随书光盘中“\图书例程\第四章\投递消息.e”





## 18) “取标记组件” 命令

命令原型为:

&lt;通用型&gt; 对象. 取标记组件 (整数型 欲寻找组件的标记数值)

本命令用于返回调用对象中具有指定标记数值文本的组件。如果不存在，将产生运行时错误。返回值会根据具体返回的数据而定。

参数<1>指定了欲寻找组件的标记数值，即该组件的“标记”属生文本的数值形式。

例如:

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
变量1	按钮			

变量1 = \_启动窗口.取标记组件 (1)

变量1.销毁 0

程序运行后会将标记为“1”的按钮销毁。

本例程源码见随书光盘中“\图书例程\第四章\取标记组件.e”

## 19) “置外形图片” 命令

命令原型为:

&lt;逻辑型&gt; 对象. 置外形图片 (通用型 图片数据, [整数型 透明颜色])

本命令用于使用指定图片来设置窗口的外形，注意图片类型不能为图标及鼠标指针。另外，图片的轮廓应该尽量简单，以免影响窗口的刷新速度。如果调用对象为窗口组件，将自动使用其所在的窗口。成功返回“真”，失败返回“假”。

参数<1>用于指定图片数据。参数值可以为JPG、GIF、BMP、DIB 等等被支持格式的图片数据、图片资源或图片文件名。

参数<2>指定了图片中透明部分的颜色。如果被省略，默认为白色。

例如: 所用图片如图4-15所示。

设定透明颜色参数为白色。

\_启动窗口.置外形图片 (#外形图片, #白色)

程序运行后会根据上图的轮廓剪裁“\_启动窗口”，运行效果如图4-16所示。

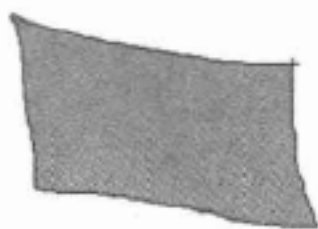


图4-15 所用图片



图4-16 窗口效果

本例程源码见随书光盘中“\图书例程\第四章\置外形图片.e”

## 20) “激活”

命令原型为:

&lt;无返回值&gt; 对象. 激活 ()





本命令用于使调用对象处于激活状态。

例如：

`_启动窗口.激活()`

程序执行后激活“\_启动窗口”。

## 21) “置托盘图标”命令

命令原型为：

`<无返回值> 对象.置托盘图标 ([通用型 图标数据], [文本型 提示信息])`

本命令用于设置本程序在系统托盘中的图标。设置后如果操作本程序的托盘图标即会触发调用本命令对象的“托盘事件”。

参数<1>指定了欲设置为托盘图标的数据。参数值可以为图标字节集数据、图标资源或图标文件名。如果省略本参数，默认为清除已有的本程序图标。

参数<2>指定了鼠标悬停在托盘图标上时显示的提示文本。本参数指定当鼠标移动到图标上后显示的提示信息。如果省略本参数，默认为空文本。

例如：

`_启动窗口.置托盘图标(#托盘图标, “托盘图标”)`

程序运行后会将“托盘图标”资源指定的图标数据设置为本程序的托盘图标，并添加提示信息。

本例程源码见随书光盘中“\图书例程\第四章\托盘事件.e”

## 22) “弹出托盘菜单”命令

命令原型为：

`<无返回值> 对象.弹出托盘菜单 (菜单 欲弹出的菜单)`

本命令用于在当前鼠标位置弹出指定菜单，本命令一般用作支持托盘菜单的弹出。

参数<1>指定了欲弹出的菜单。指定菜单内必须包含有子菜单，且所处窗口必须与调用对象窗口一致。

例如：

`_启动窗口.弹出托盘菜单(易语言菜单)`

程序执行后会在当前鼠标位置弹出“易语言菜单”的子菜单项。

本例程源码见随书光盘中“\图书例程\第四章\托盘事件.e”

## 23) “置父窗口”命令

命令原型为：

`<无返回值> 对象.置父窗口 (通用型 父窗口或窗口组件)`

本命令用于设置指定窗口或窗口组件为本对象的父窗口。设置后，若父窗口被销毁，那么调用的对象也将会被一起销毁。

**注解：**本命令执行后可能会使某些属性动态设置失效，例如“左边”、“顶边”等，这时可以使用“移动”命令来实现所需要的效果。





参数<1>指定了将成为调用对象父窗口的窗口或窗口组件。

例如：

窗口1.置父窗口 (启动窗口)

窗口1.移动 (0, 0, , )

程序运行后将“窗口1”置为“\_启动窗口”的子窗口；移动“窗口1”到“\_启动窗口”内的左上角。

本例程源码见随书光盘中“\图书例程\第四章\置父窗口.e”

## 4.2 菜单

菜单让程序的操作变得简单、方便，也使程序的界面更加美观。在易语言中制作一组实用、美观的菜单是非常简单的事情。

### 4.2.1 创建菜单

为程序创建菜单，需要使用易语言所提供的“菜单编辑器”。选择易语言主菜单“工具”→“菜单编辑器”；或在窗体上单击鼠标右键，选择右键菜单中的“菜单编辑器”；还可以使用热键[Ctrl+E]调出菜单编辑器。

在菜单编辑器对话框中，“向前插入”和“向后插入”按钮用来插入新的菜单项，“左移←”和“右移→”按钮可以调整菜单的层次。顶层菜单必须有子菜单，否则会出现错误。可以点击“上移↑”和“下移↓”按钮来调整菜单的显示先后顺序。

下面具体讲解如何新建菜单。

新建一个易程序，打开菜单编辑器，在编辑器中新建3个顶层菜单“程序”、“编辑”、“帮助”，然后给每个顶层菜单加入子菜单，如图4-17所示。

可以试运行程序，查看新建的菜单，如图4-18所示。

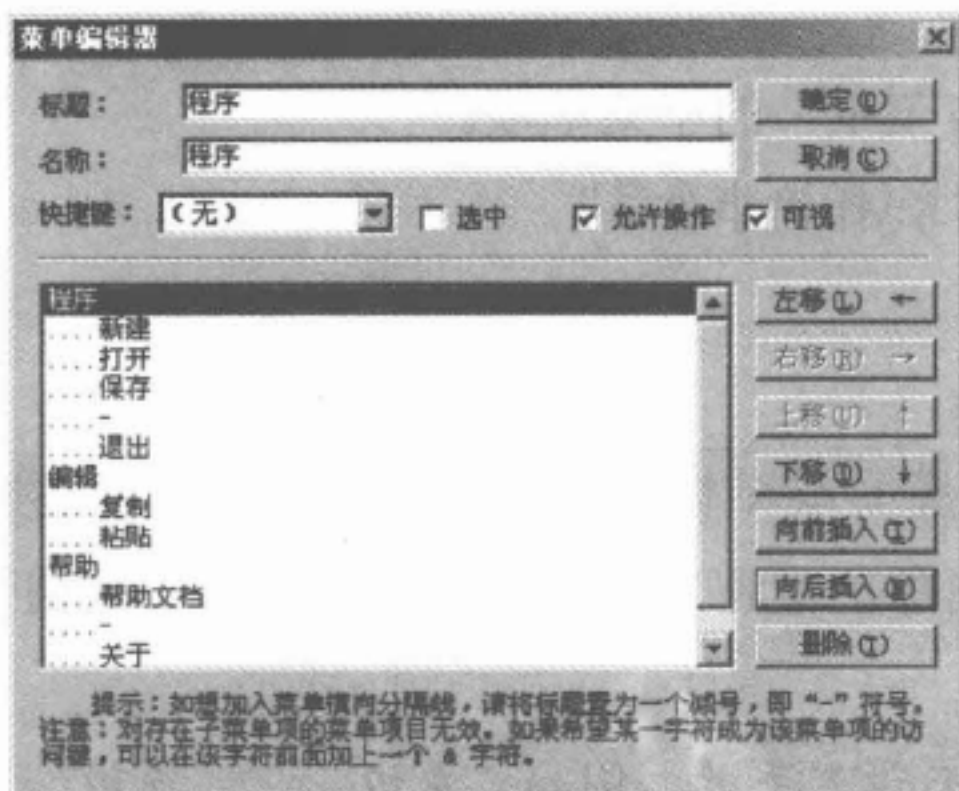


图4-17 编辑菜单

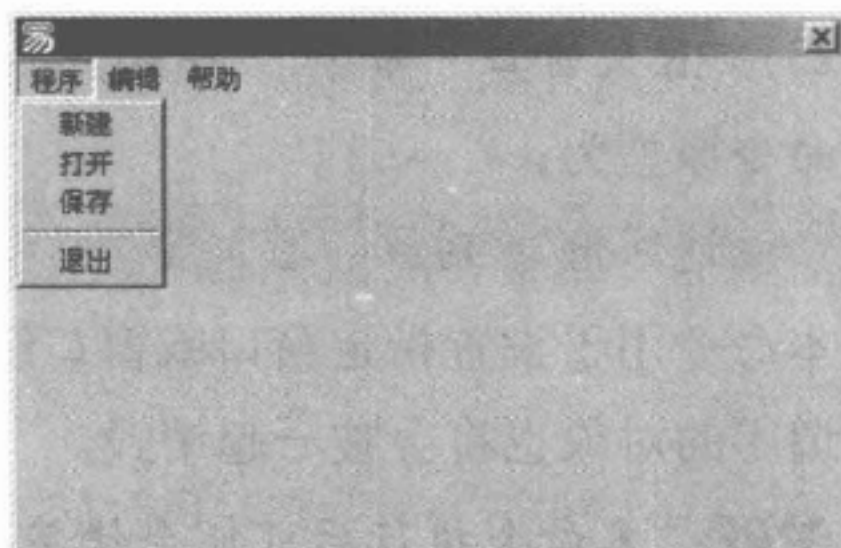


图4-18 菜单效果





菜单中“保存”和“退出”选项之间的分隔线，是菜单项“-”。

## 4.2.2 菜单的热键及属性

### 1) 菜单的热键

可以为菜单项设置快捷键。即按住Alt键后，同时按下另一个键，就可以调用指定的菜单项功能。一般情况下，有两种表示方法：菜单名（&字母）或&字母.菜单名

例如：将刚才编辑过的菜单都加上热键，如图4-19所示。

运行程序后，可以使用自定义的热键来打开菜单，同时按下Alt和F键，效果如图4-20所示。

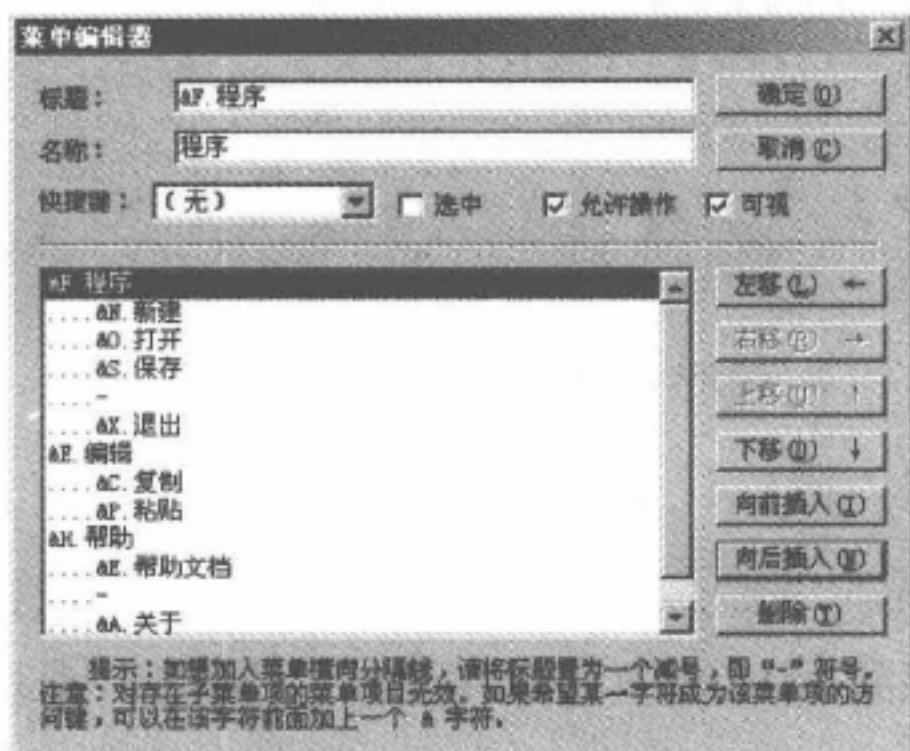


图4-19 菜单加热键

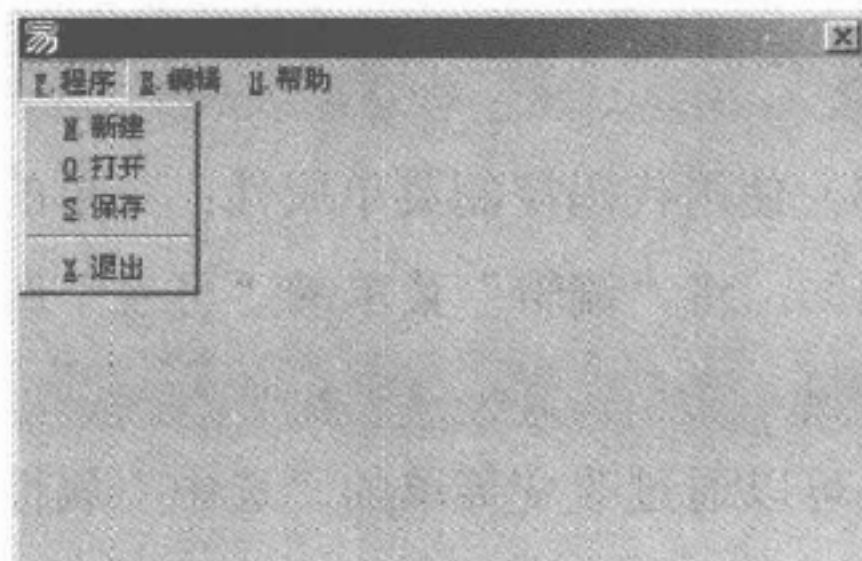


图4-20 用热键打开菜单

### 2) 菜单的属性

在编辑菜单项时会发现，菜单有“选中”、“允许操作”及“可视”这3个属性。

(1) “选中”属性：可以控制菜单是否为可选菜单，一个菜单“选中”属性被选择后，该菜单的前面就会出现“√”，如图4-21所示，图中的“关于”菜单的“选中”属性被选择。

(2) “允许操作”属性：菜单的“允许操作”属性类似组件的“禁止”属性，当菜单的“允许操作”属性被取消选择后，菜单就变成灰色，将不能对该菜单进行任何操作，如图4-22所示，图中的“程序”菜单的“允许操作”属性没有选择。

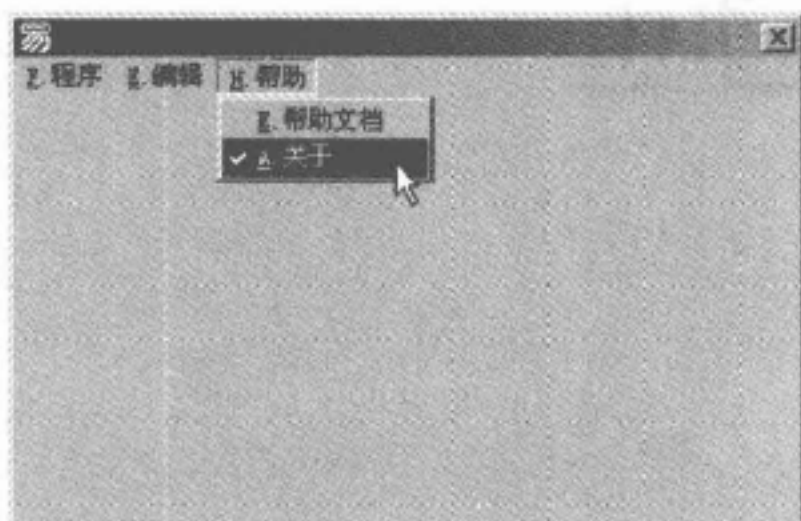


图4-21 菜单的“选中”属性

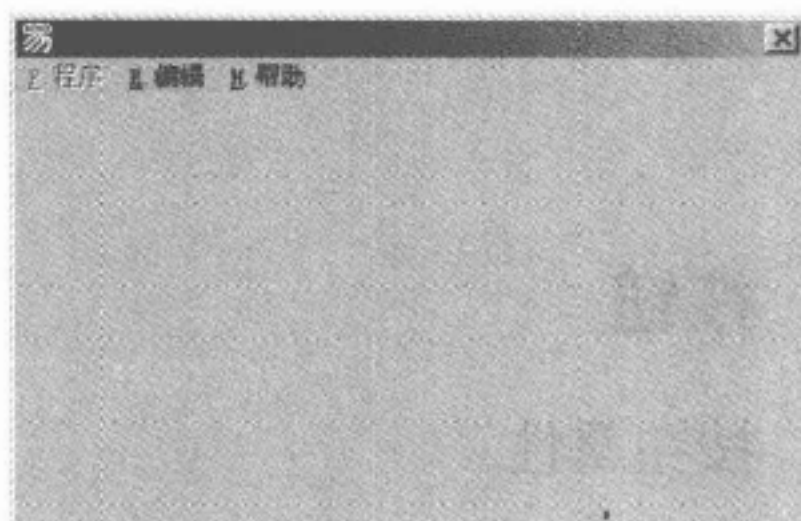


图4-22 菜单的“允许操作”属性

(3) “可视”属性：当菜单的“可视”属性被取消选择后，该菜单项就会隐藏，程序





运行过程中不可见，如图4-23所示，其中的“编辑”菜单被取消“可视”，所以运行后看不到该菜单项。

(4) 菜单被选择事件：要实现菜单上各选项的功能，就需要在该项菜单被选择事件子程序中加入相应的代码。

在窗口中点击菜单中的一个选项，就会自动生成该项被选择事件子程序。

例如：给刚才编辑过的菜单中的“退出”选项，加入结束程序运行的功能。需要在菜单中选择“退出”选项，然后在生成的“\_退出\_被选择”子程序中输入代码。

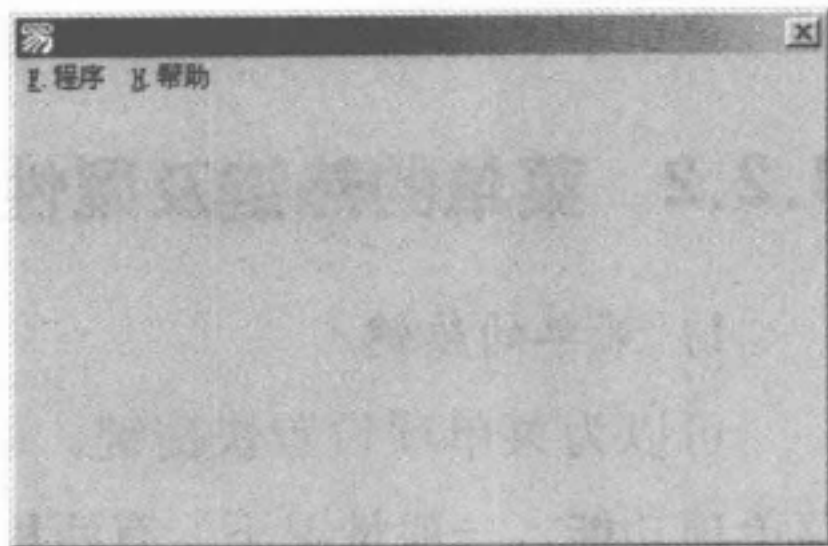


图4-23 菜单的“可视”属性

子程序名	返回值类型	公开	备注
_退出_被选择			

结束 ()

(5) 使用代码控制菜单属性：菜单的属性可以在程序中用代码进行控制。

例如：将“编辑”菜单的“可视”属性设置为假。

编辑. 可视=假

还可以通过改变菜单的“选中”属性，让该菜单项选中和取消选中。例如，刚才编辑过的菜单，点击菜单中的“关于”选项，在“\_关于\_被选择”子程序中输入代码：

子程序名	返回值类型	公开	备注
_关于_被选择			

```
--- 如果 (关于.选中 = 真)
--- 关于.选中 = 假
--- 关于.选中 = 真
```

当程序运行后，就可以显示和取消“关于”前的“√”。

本例程源码见随书光盘中“\图书例程\第四章\菜单编辑.e”。

## 4.3 按钮类组件

### 4.3.1 按钮

#### 4.3.1.1 按钮属性

##### 1) “图片”属性

属性类型：字节集；有效范围：设计时、编程时；编程时权限：使用、读取、更改





本属性用于指定显示在按钮上的图片。显示图片后按钮的标题将被隐藏。

例如：

按钮\_图片.图片 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)

程序执行后会读入 “C:\WINDOWS\Prairie Wind.bmp” (假设文件存在) 作为 “按钮\_图片” 的图片。

## 2) “类型” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定按钮的类型。当在窗口上按下Enter键时，如果没有将窗口的“回车下移输入焦点”属性设置为真，则等同于按下了具有“默认”类型的按钮。因此具有“默认”类型的按钮在同一窗口上应当只有一个。“通常”类型的按钮无对应的操作默认键。

属性可以是以下值之一：0（通常）；1（默认）。

例如：

按钮\_类型.类型 = 1

程序执行后将“按钮\_类型”的类型设为默认型。

## 3) “标题” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定按钮所显示的标题文字。标题只有在图片为空时才有效（如果按钮具有图片数据，将会覆盖按钮的标题），如果希望某一字符成为此按钮的访问键，可以在该字符前面加上一个“&”字符。

例如：

按钮\_标题.标题 = “易语言”

程序执行后将“按钮\_标题”的标题设为“易语言”。

## 4) “横向对齐方式” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置按钮内标题文字的横向对齐方式。

本属性可以是以下值之一：0（左边）；1（居中）；2（右边）。

例如：

按钮\_横向对齐.横向对齐方式 = 2

程序执行后将“按钮\_横向对齐”标题文字的横向对齐方式设为右对齐。

## 5) “纵向对齐方式” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置按钮内标题文字的纵向对齐方式。

本属性可以是以下值之一：0（顶边）；1（居中）；2（底边）。





例如：

按钮\_纵向对齐.纵向对齐方式 = 2

程序执行后将“按钮\_纵向对齐”标题文字的纵向对齐方式设为底对齐。

#### 6) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置按钮内标题文字所使用的字体。只有在图片属性为空时才有效。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

变量1.加粗 = 真

变量1.字体名称 = “黑体”

按钮1.字体 = 变量1

程序执行后会为“按钮1”中的标题文字设置一个新的字体。

#### 4.3.1.2 按钮事件

“被单击”事件原型为：

无返回值，无参数

当单击按钮后即产生本事件。

例如：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

» 输出调试文本（“按钮被单击”）

当“按钮1”被单击时在输出面板中显示“按钮被单击”。

### 4.3.2 图形按钮

图形按钮一般用于组建个性化的程序用户界面时使用。

#### 4.3.2.1 图形按钮属性

##### 1) “类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定图形按钮的显示样式。

本属性可以是以下值之一：0（按钮）；1（选择框）。

例如：

图形按钮1.类型 = 1

图形按钮1.选中 = 真

图形按钮1.正常图片 = 读入文件（“C:\a.bmp”）

图形按钮1.按下图片 = 读入文件（“C:\c.bmp”）





```
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
图形按钮1.透明颜色 = #黑色
```

程序执行后将“图形按钮1”的显示样式设为选择框式；设置当前状态为选中；设置正常状态时显示的图片为“C:\a.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。设置忽略图片中的黑色像素。

## 2) “选中”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
当类型为选择框时，本属性为当前选中状态。选中为“真”，未选中为“假”。

例如：

```
图形按钮1.类型 = 1
图形按钮1.选中 = 真
图形按钮1.正常图片 = 读入文件 ("C:\a.bmp")
图形按钮1.按下图片 = 读入文件 ("C:\c.bmp")
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
图形按钮1.透明颜色 = #黑色
```

程序执行后将“图形按钮1”的显示样式设为选择框式；设置当前状态为选中；设置正常状态时显示的图片为“C:\a.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。设置忽略图片中的黑色像素。

## 3) “正常图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定在正常情况下所显示图片（支持BMP、JPG、GIF格式的图片数据）。

例如：

```
图形按钮1.类型 = 0
图形按钮1.正常图片 = 读入文件 ("C:\a.bmp")
图形按钮1.点燃图片 = 读入文件 ("C:\b.bmp")
图形按钮1.按下图片 = 读入文件 ("C:\c.bmp")
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
```

程序执行后将“图形按钮1”的显示样式设为按钮式；设置正常状态时显示的图片为“C:\a.bmp”；设置点燃状态时显示的图片为“C:\b.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。

## 4) “点燃图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

当类型为按钮时，本属性用于指定当鼠标移动到按钮上时所显示图片（支持BMP、JPG、GIF格式的图片数据）。





例如：

```
图形按钮1.类型 = 0  
图形按钮1.正常图片 = 读入文件 ("C:\a.bmp")  
图形按钮1.点燃图片 = 读入文件 ("C:\b.bmp")  
图形按钮1.按下图片 = 读入文件 ("C:\c.bmp")  
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
```

程序执行后将“图形按钮1”的显示样式设为按钮式；设置正常状态时显示的图片为“C:\a.bmp”；设置点燃状态时显示的图片为“C:\b.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。

#### 5) “按下图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当选择框被选中或者按钮被按下后所显示图片（支持BMP、JPG、GIF格式的图片数据）。

例如：

```
图形按钮1.类型 = 0  
图形按钮1.正常图片 = 读入文件 ("C:\a.bmp")  
图形按钮1.点燃图片 = 读入文件 ("C:\b.bmp")  
图形按钮1.按下图片 = 读入文件 ("C:\c.bmp")  
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
```

程序执行后将“图形按钮1”的显示样式设为按钮式；设置正常状态时显示的图片为“C:\a.bmp”；设置点燃状态时显示的图片为“C:\b.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。

#### 6) “禁止图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当按钮被禁止后所显示图片（支持BMP、JPG、GIF格式的图片数据）。

例如：

```
图形按钮1.类型 = 0  
图形按钮1.正常图片 = 读入文件 ("C:\a.bmp")  
图形按钮1.点燃图片 = 读入文件 ("C:\b.bmp")  
图形按钮1.按下图片 = 读入文件 ("C:\c.bmp")  
图形按钮1.禁止图片 = 读入文件 ("C:\d.bmp")
```

程序执行后将“图形按钮1”的显示样式设为按钮式；设置正常状态时显示的图片为“C:\a.bmp”；设置点燃状态时显示的图片为“C:\b.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。

#### 7) “透明颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定所有图片中透明部分的颜色。如果为“#默认色”常量值，则表示无透明色。

例如：

```
图形按钮1.类型 = 1
图形按钮1.选中 = 真
图形按钮1.正常图片 = 读入文件 (“C:\a.bmp”)
图形按钮1.按下图片 = 读入文件 (“C:\c.bmp”)
图形按钮1.禁止图片 = 读入文件 (“C:\d.bmp”)
图形按钮1.透明颜色 = #黑色
```

程序执行后将“图形按钮1”的显示样式设为按钮式；设置正常状态时显示的图片为“C:\a.bmp”；设置按下状态时显示的图片为“C:\c.bmp”；设置禁止状态显示的图片为“C:\d.bmp”。

4.3.2.2 图形按钮事件

“被单击”事件原型为：

返回值类型：无返回值；无参数

程序运行后，当单击图形按钮后即产生此事件。

例如：

子程序名	返回值类型	公开	备注
_图形按钮1_被单击			

» 输出调试文本 (“被单击”)

程序运行后当单击“图形按钮1”在输出面板中显示一段文字。

4.4 列表类组件

4.4.1 组合框

4.4.1.1 组合框属性

1) “类型”属性

属性类型：整数型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于设置组合框的样式。属性可以是以下值之一：0（可编辑列表式）；1（可编辑下拉式）；2（不可编辑下拉式）。

例如：

```
组合框_类型.类型 = 2
```

程序执行后将“组合框\_类型”的样式设为不可编辑下拉式。





## 2) “内容”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在下拉列表框编辑部分中的文本内容。当类型为“不可编辑下拉式”时，本属性无效。

例如：

```
组合框_内容.内容 = “易语言”
```

程序执行后在“组合框\_内容”的编辑部分显示“易语言”。

## 3) “最大文本长度”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在下拉列表框的编辑部分所允许输入的最大文本长度，为0表示没有限制。当类型为“不可编辑下拉式”时，本属性无效。

例如：

```
组合框_长度.最大文本长度 = 10
```

程序执行后指定“组合框\_长度”编辑部分最大长度为10个字节。

## 4) “起始选择位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于获取或设置所选择文本的起始点；0为位置1，1为位置2，以此类推。如果没有文本被选中，则指出光标位置。如果设置位置时使用值-1，则将当前光标位置移动到文本尾部。当类型为“不可编辑下拉式”时，本属性无效。

例如：

```
组合框1.内容 = “易语言”
```

```
组合框1.起始选择位置 = 3
```

```
组合框1.被选择字符数 = 2
```

```
» 输出调试文本 (组合框1.被选择文本)
```

程序执行后将“组合框1”的编辑部分内容设置为“易语言”；设置起始选择位置为3；选择从第3位置开始（包含第3位置）2个字节；将被选择的文字显示在输出面板中。

## 5) “被选择字符数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于获取或设置所选择的字符数。如果设置字符数时使用值-1，则选择所有文本。当类型为“不可编辑下拉式”时，本属性无效。

例如：

```
组合框1.内容 = “易语言”
```

```
组合框1.起始选择位置 = 3
```

```
组合框1.被选择字符数 = 2
```

```
» 输出调试文本 (组合框1.被选择文本)
```





程序执行后将“组合框1”的编辑部分内容设置为“易语言”；设置起始选择位置为3；选择从第3位置开始（包含第3位置）2个字节；将被选择的文字显示在输出面板中。

6) “被选择文本” 属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取、更改

本属性用于获取或替换当前在下拉列表框编辑部分中所选择的文本。当类型为“不可编辑下拉式”时，本属性无效。

例如：

```
组合框1.内容 = “易语言”
组合框1.起始选择位置 = 3
组合框1.被选择字符数 = 2
» 输出调试文本 (组合框1.被选择文本)
```

程序执行后将“组合框1”的编辑部分内容设置为“易语言”；设置起始选择位置为3；选择从第3位置开始（包含第3位置）2个字节；将被选择的文字显示在输出面板中。

7) “自动排序” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否开启组合框项目间的自动排序功能。“真”为开启，“假”为关闭。排序方式为按项目文本依次比对ASCII码升序排列。

例如：

```
组合框_排序.自动排序 = 真
程序执行后打开“组合框_排序”的自动排序功能。
```

8) “行间距” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定各列表项目行之间的间距尺寸。单位为像素。

例如：

```
组合框_行间距.行间距 = 10
程序执行后将“组合框_行间距”的项目之间的距离设为10像素。
```

9) “文本颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定组合框内各项目的文本颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

变量名	类型	静态	数组	备注
变量1	字体			





```

组合框1.文本颜色 = #黄色
组合框1.背景颜色 = #黑色
变量1.字体大小 = 25
变量1.倾斜 = 真
组合框1.字体 = 变量1

```

程序执行后会将“组合框1”项目的文字颜色设为黄色；背景颜色设为黑色；改变组合框项目字体为新字体。

#### 10) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定组合框中项目的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```

组合框1.文本颜色 = #黄色
组合框1.背景颜色 = #黑色
变量1.字体大小 = 25
变量1.倾斜 = 真
组合框1.字体 = 变量1

```

程序执行后会将“组合框1”项目的文字颜色设为黄色；背景颜色设为黑色；改变组合框项目字体为新字体。

#### 11) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定组合框内文字的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```

组合框1.文本颜色 = #黄色
组合框1.背景颜色 = #黑色
变量1.字体大小 = 25
变量1.倾斜 = 真
组合框1.字体 = 变量1

```

程序执行后会将“组合框1”项目的文字颜色设为黄色；背景颜色设为黑色；改变组合框项目字体为新字体。

#### 12) “现行选中项”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定下拉列表框中现行被选中的列表项目的位置，位置值从 0 开始，-1 表示现行没有被选中的列表项。

例如：

输出调试文本 (组合框1.现行选中项)

程序执行后在输出面板中显示“组合框1”当前“现行选中项”的属性值。

### 13) “列表项目”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定各列表项目。

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-24所示。

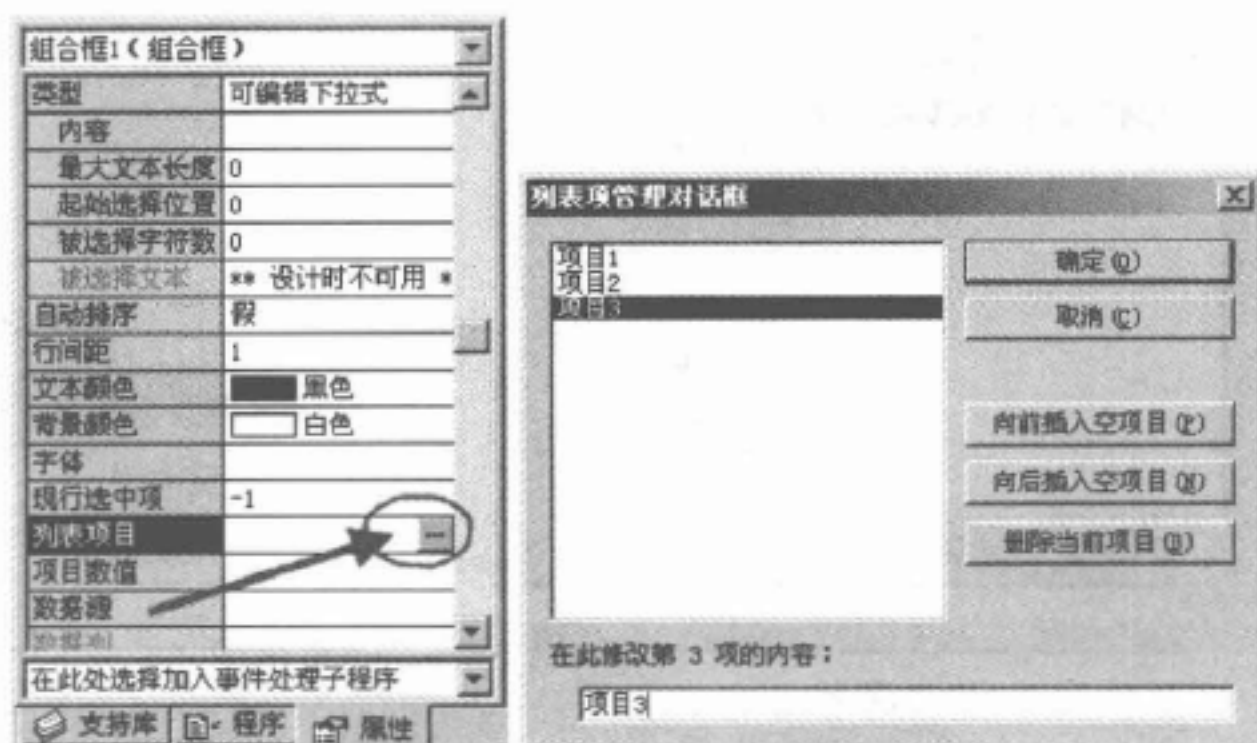


图4-24 打开项目编辑器

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。

本属性数据存储格式见表4-1。

表4-1 数据存储格式

数据类型 (长度)	说明	数据类型 (长度)	说明
短整型	项目数	整型	第二个项目文字长度 (字节数)
整型	第一个项目文字长度 (字节数)	字节型	第二个项目文字的ASCII码数组
字节型	第一个项目文字的ASCII码数组	.....	.....

例如：

```
组合框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98, 3, 0, 0, 0, 99, 99, 99 }  
组合框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }  
组合框1.现行选中项 = 2  
输出调试文本 (组合框1.取项目数值 (组合框1.现行选中项))
```

程序执行后改变“组合框1”的列表项目为a, bb, ccc；改变项目数值为1, 2, 3；设置现行选择项为2，既选择第3个项目；在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\组合框\列表项目.e”



**注解：**编程时动态地改变本属性和使用“加入项目”、“删除项目”等命令比优势在于：在大数据量修改时，速度会快很多。

### 14) “项目数值”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定与各列表项目相关联的数值，该数值在程序中可以取出或修改。  
在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-25所示。

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。

本属性数据格式见表4-2。

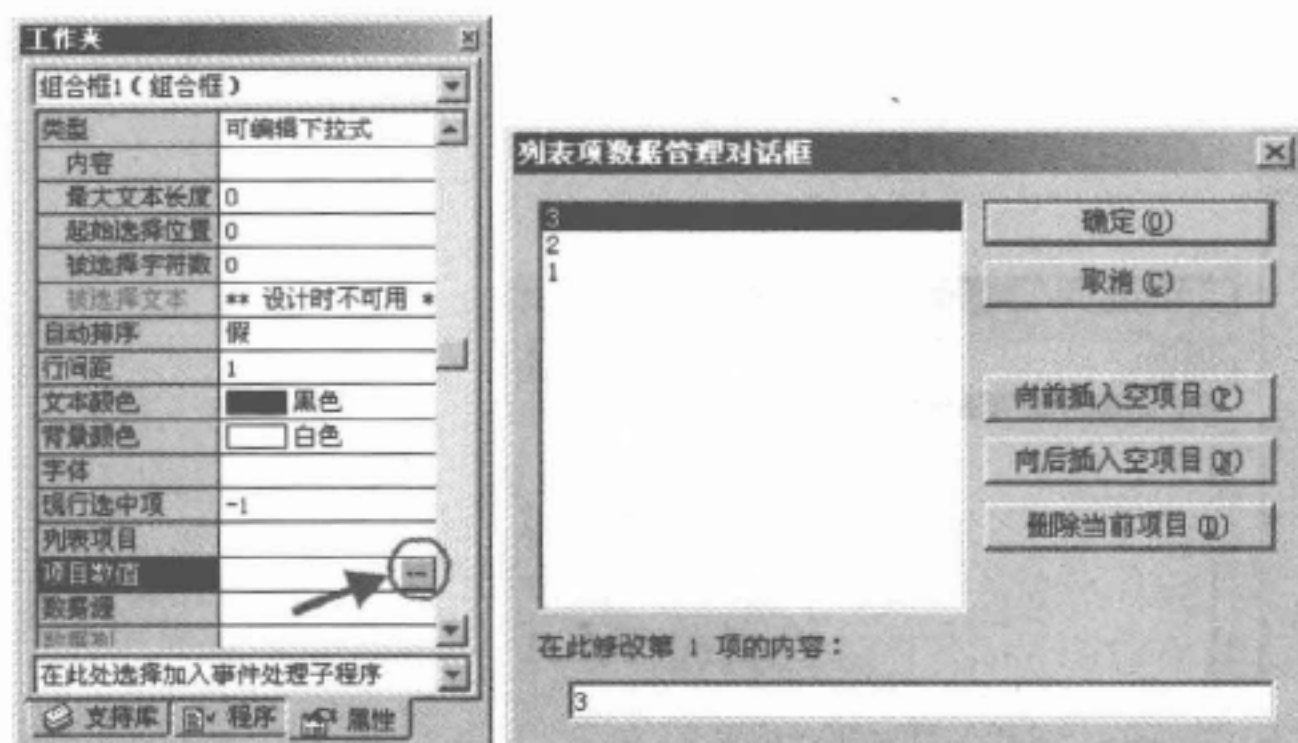


图4-25 打开项目数值编辑器

表4-2 数据格式

数据类型 (长度)	说明
整数型	项目一对应的数值
整数型	项目二对应的数值
.....	.....

例如：

组合框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98, 3, 0, 0, 0, 99, 99, 99 }

组合框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }

组合框1.现行选中项 = 2

» 输出调试文本 (组合框1.取项目数值 (组合框1.现行选中项))

程序执行后改变“组合框1”的列表项目为a, bb, ccc；改变项目数值为1, 2, 3；设置现行选择项为2，既选择第3个项目；在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\组合框\列表项目.e”

### 15) “数据源”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

组合框1.数据源 = “数据源1”

组合框1.数据列 = “city”





程序执行后将“组合框1”的数据来源和“数据源1”绑定；将组合框显示内容与数据源当前记录中“city”列绑定。

16) “数据列” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```

组合框1.数据源 = “数据源1”
组合框1.数据列 = “city”
    
```

程序执行后将“组合框1”的数据来源和“数据源1”绑定；将组合框显示内容与数据源当前记录中“city”列绑定。

4.4.1.2 组合框事件

1) “列表项被选择” 事件

事件原型为：

无返回值，无参数

当组合框的表项被选择后，触发本事件。

例如：

子程序名	返回值类型	公开	备 注
_组合框1_列表项被选择			

» 输出调试文本 (组合框1.内容)

程序执行后，当“组合框1”的列表项被选择后在输出面板中显示当前的内容。

2) “编辑内容被改变” 事件

事件原型为：

无返回值，无参数

当程序运行后手动改变组合框的内容时触发本事件。当组合框类型为“不可编辑下拉式”时本事件无效。

例如：

子程序名	返回值类型	公开	备 注
_组合框1_编辑内容被改变			

» 输出调试文本 (组合框1.内容)

程序执行后，当“组合框1”的内容被手动改变后，在输出面板显示当前“组合框1”的内容。





## 3) “将弹出列表”事件

事件原型为:

无返回值; 无参数

在将弹出组合框的列表部分之前产生此事件。在处理此事件时, 可以即时填充组合框中的列表项目。当类型为“可编辑列表式”时本事件无效。

例如:

子程序名	返回值类型	公开	备注
_组合框1_将弹出列表			

» 输出调试文本 (“将弹出列表”)

程序运行后, 当“组合框1”弹出下拉列表前在输出面板中显示“将弹出列表”。

## 4) “列表被关闭”事件

事件原型为:

无返回值; 无参数

在关闭被弹出的组合框列表部分后产生此事件。当类型为“可编辑列表式”时本事件无效。

例如:

子程序名	返回值类型	公开	备注
_组合框1_列表被关闭			

» 输出调试文本 (“列表被关闭”)

程序运行后, 当“组合框1”下拉列表被关闭时在输出面板中显示“列表被关闭”。

## 5) “双击选择”事件

事件原型为:

无返回值; 无参数

当使用鼠标左键在某列表项目上双击时产生此事件。只有当类型为“可编辑列表式”时本事件有效。

例如:

子程序名	返回值类型	公开	备注
_组合框1_双击选择			

» 输出调试文本 (“双击选择”)

程序运行后, 当双击“组合框1”的列表项时在输出面板中显示“双击选择”。

## 4.4.1.3 组合框命令

## 1) “取顶端可见项目”命令

命令原型为:



### <整数型> 对象. 取顶端可见项目 ()

本命令用于获取组合框列表部分中当前最顶端可见项目的索引。0 为项目一，1 为项目二，以此类推。失败返回 -1。本命令一般用于“可编辑列表式”的组合框列表项较多时。

例如：

#### 输出调试文本 (组合框1.取顶端可见项目 ())

程序执行后在输出面板中显示“组合框1”当前最顶端可见项目的索引。

### 2) “置顶端可见项目”命令

命令原型为：

### <逻辑型> 对象. 置顶端可见项目 (整数型 项目索引)

本命令用于设置组合框列表部分中当前最顶端的可见项目，必要时将自动滚动组合框的列表部分。成功返回“真”，失败返回“假”。本命令一般用于“可编辑列表式”的组合框列表项较多时。

参数<1>指定了欲设置为顶端的项目索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

例如：

#### 组合框1.置顶端可见项目 (1)

程序执行后将索引为“1”的列表项置为“组合框1”中可见的第一条项目。

### 3) “取项目数”命令

命令原型为：

### <整数型> 对象. 取项目数 ()

本命令用于获取组合框中列表项目的数量。

例如：

#### 输出调试文本 (组合框1.取项目数 ())

程序执行后在输出面板中显示“组合框1”的列表项目数量。

### 4) “取项目数值”命令

命令原型为：

### <整数型> 对象. 取项目数值 (整数型 项目索引)

本命令用于获取与指定项目相关联的数值。如果指定项目不存在，将返回 -1。

参数<1>指定了欲获取项目数值的项目索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

例如：

#### 输出调试文本 (组合框1.取项目数 (1))

程序执行后在输出面板中显示“组合框1”第二个项目所对应的项目数值。





## 5) “置项目数值”命令

命令原型为:

&lt;逻辑型&gt; 对象. 置项目数值 (整数型 项目索引, 整数型 欲置入的项目数值)

本命令用于设置与指定项目相关联的数值。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置项目数值的项目索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

参数&lt;2&gt;用于指定项目对应的新数值。

例如:

```
组合框1.置项目数值 (2, 123)
» 输出调试文本 (组合框1.取项目数值 (2))
```

程序执行后设置“组合框1”的第三个项目对应的数值为“123”；在输出面板中显示“组合框1”的第三个项目对应的数值。

## 6) “取项目文本”命令

命令原型为:

&lt;文本型&gt; 对象. 取项目文本 (整数型 项目索引)

本命令用于获取指定项目的文本。如果指定项目不存在，将返回空文本。

参数<1>指定了欲获取项目文本的项目索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

例如:

```
组合框1.清空 0
组合框1.加入项目 (“易语言”, 123)
组合框1.插入项目 (0, “ABC”, 321)
组合框1.置项目文本 (0, “组合框”)
» 输出调试文本 (组合框1.取项目文本 (0))
组合框1.选择 (“易语言”)
```

程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

## 7) “置项目文本”命令

命令原型为:

&lt;逻辑型&gt; 对象. 置项目文本 (整数型 项目索引, 文本型 欲置入的项目文本)

本命令用于设置指定项目的文本。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置项目文本的项目索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。



参数<2>指定了列表项的新文本。

例如：

```

组合框1.清空 0
组合框1.加入项目 (“易语言”, 123)
组合框1.插入项目 0, “ABC”, 321)
组合框1.置项目文本 0, “组合框”)
» 输出调试文本 (组合框1.取项目文本 0))
组合框1.选择 (“易语言”)

```

程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

#### 8) “加入项目”命令

命令原型为：

<整数型> 对象. 加入项目 (文本型 欲加入项目的文本, [整数型 与欲加入项目相关的数值])

本命令用于加入指定项目到组合框列表部分的尾部，成功返回加入后该项目所处的位置，失败返回-1。

参数<1>指定了新项目的文本。

参数<2>指定了与新项相关的数值，如果省略本参数，默认值为0。

例如：

```

组合框1.清空 0
组合框1.加入项目 (“易语言”, 123)
组合框1.插入项目 0, “ABC”, 321)
组合框1.置项目文本 0, “组合框”)
» 输出调试文本 (组合框1.取项目文本 0))
组合框1.选择 (“易语言”)

```

程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

#### 9) “插入项目”命令

命令原型为：

<整数型> 对象. 插入项目 (整数型 欲插入的位置, 文本型 欲插入项目的文本, [整数型 与欲插入项目相关的数值])

本命令用于插入指定项目到组合框列表部分的指定位置处，成功返回插入后该项目所处的位置，失败返回-1。





参数<1>指定了新项欲插入的位置。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

参数<2>指定了新项目的文本。

参数<3>指定了与新项相关的数值，如果省略本参数，默认值为 0。

例如：

```

组合框1.清空 ()
组合框1.加入项目 ("易语言", 123)
组合框1.插入项目 (0, "ABC", 321)
组合框1.置项目文本 (0, "组合框")
» 输出调试文本 (组合框1.取项目文本 (0))
组合框1.选择 ("易语言")
  
```

程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

#### 10) “删除项目”命令

命令原型为：

<逻辑型> 对象. 删除项目 (整数型 项目索引)

本命令用于删除组合框列表部分指定位置处的项目。成功返回“真”，失败返回“假”。

参数<1>指定了欲删除项目的索引。项目索引从0开始，0 为项目一，1 为项目二，以此类推。

例如：

```

组合框1.删除项目 (0)
  
```

程序执行后，删除“组合框1”中的第一个项目。

#### 11) “清空”命令

命令原型为：

<无返回值> 对象. 清空 ()

本命令用于删除组合框列表部分中的所有项目。

例如：

```

组合框1.清空 ()
组合框1.加入项目 ("易语言", 123)
组合框1.插入项目 (0, "ABC", 321)
组合框1.置项目文本 (0, "组合框")
» 输出调试文本 (组合框1.取项目文本 (0))
组合框1.选择 ("易语言")
  
```





程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

#### 12) “选择”命令

命令原型为：

<整数型> 对象. 选择 (文本型 欲选择的项目文本)

本命令用于在所有项目中寻找首部包含指定文本的项目，如找到，则选中它，并返回该项目的位置索引，否则返回 -1。

参数<1>指定了选择时的查找条件。

例如：

```
组合框1.清空 0
组合框1.加入项目 (“易语言”, 123)
组合框1.插入项目 (0, “ABC”, 321)
组合框1.置项目文本 (0, “组合框”)
» 输出调试文本 (组合框1.取项目文本 (0))
组合框1.选择 (“易语言”)
```

程序执行后将“组合框1”中的所有项目删除；加入文本为“易语言”，数值为“123”的新项目；在第一个项目前插入文本为“ABC”，数值为“321”的新项目；将第一个项目的文本改为“组合框”；在输出面板中显示第一个项目的文本；查找文本为“易语言”的项目并选择它。

## 4.4.2 列表框

### 4.4.2.1 列表框属性

#### 1) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置列表框边框的样式。样式值可以是：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
列表框_边框.边框 = 5
```

程序执行后将“列表框\_边框”的边框改为单线边框。

#### 2) “自动排序”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否开启列表框项目间的自动排序功能。“真”为开启，“假”为关闭。排序方式为按项目文本依次比对ASCII码升序排列。





例如：

```
列表框_自动排序.自动排序 = 真
```

程序执行后打开“列表框\_自动排序”的自动排序功能。

### 3) “多列”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定列表框在项目较多时是否多列来显示。“真”为使用多列，“假”为不使用。

例如：

```
列表框_多列.多列 = 真
```

程序执行后开启“列表框\_多列”的多列显示功能。

### 4) “行间距”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定各列表项目行之间的间距尺寸。单位为像素。

例如：

```
列表框_行间距.行间距 = 10
```

程序执行后将“列表框\_行间距”的项目之间的距离设为10像素。

### 5) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置列表框内文本显示的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
列表框_颜色.文本颜色 = #绿色
```

程序执行后会将“列表框\_颜色”的文本颜色设为绿色。

### 6) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置列表框内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
列表框_颜色.背景颜色 = #黑色
```

程序执行后会将“列表框\_颜色”的背景颜色设为黑色。

### 7) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置列表框内文字所使用的字体。





例如：

变量名	类型	静态	数组	备注
变量1	字体			

变量1. 字体大小 = 30  
变量1. 字体名称 = “幼圆”  
变量1. 加粗 = 真  
列表框1. 字体 = 变量1

程序执行后会为“列表框1”中的文字设置一个新的字体。

8) “允许选择多项” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否允许用户同时选择列表框中的多个列表项。

例如：

列表框\_多项.允许选择多项 = 真

程序执行后开启“列表框\_多项”同时选择多项功能。

9) “现行选中项” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性指定列表框中现行被选中列表项目的位置，位置值从0开始，-1表示现行没有被选中的列表项。如果“允许选择多项”属性为真，则本属性无效，其值恒为 -1。

例如：

输出调试文本(列表框1.现行选中项)

程序执行后，在输出面板中显示“列表框1”的现行选中项。

10) “列表项目” 属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定各列表项目。

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-26所示。

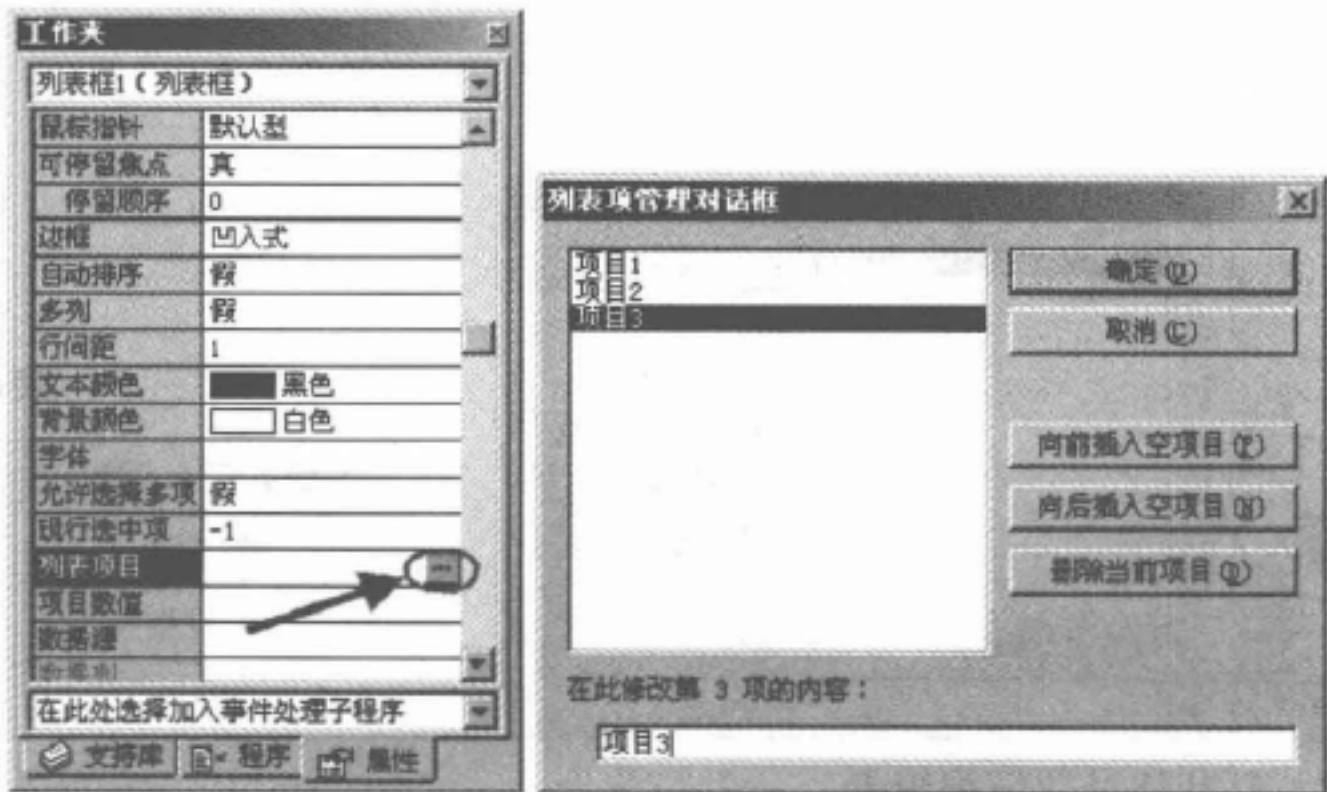


图4-26 打开项目编辑器





本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。

本属性数据存储格式见表4-3。

表4-3 数据存储格式

数据类型（长度）	说明	数据类型（长度）	说明
短整数型	项目数	整数型	第二个项目文字长度（字节数）
整数型	第一个项目文字长度（字节数）	字节型	第二个项目文字的ASCII码数组
字节型	第一个项目文字的ASCII码数组	.....	.....

例如：

```
列表框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98, 3, 0, 0, 0, 99, 99, 99 }
```

```
列表框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }
```

```
列表框1.现行选中项 = 2
```

```
» 输出调试文本 (列表框1.取项目数值 (列表框1.现行选中项))
```

程序执行后改变“列表框1”的列表项目为a, bb, ccc; 改变项目数值为1, 2, 3; 设置现行选择项为2, 既选择第3个项目; 在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\列表框\列表项目.e”

#### 11) “项目数值”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定与各列表项目相关联的数值，该数值在程序中可以取出或修改。

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-27所示。

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。

本属性数据见表4-4。

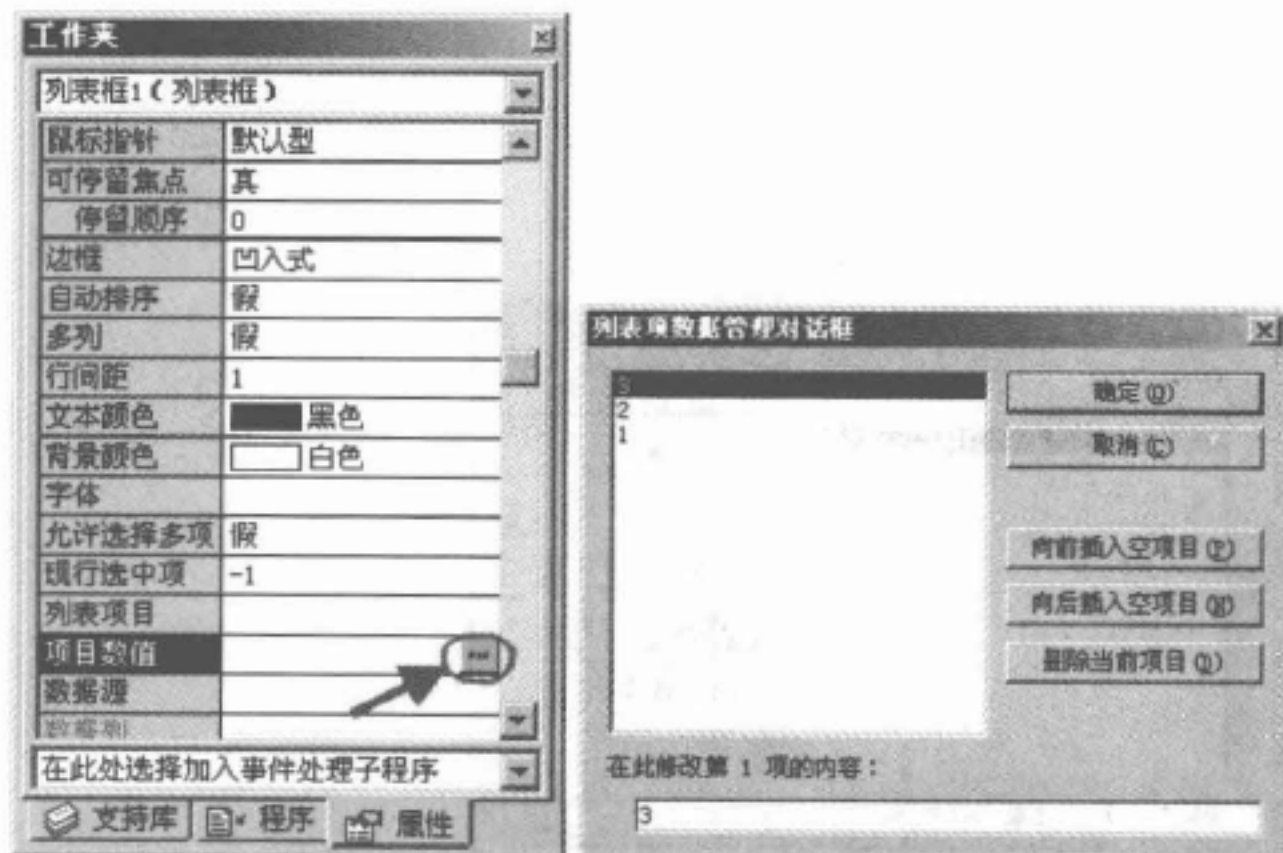


图4-27 打开项目数值编辑器

表4-4

数据类型（长度）	说明
整数型	项目一对应的数值
整数型	项目二对应的数值
.....	.....





例如：

```
列表框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98,
, 3, 0, 0, 0, 99, 99, 99 }
列表框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }
列表框1.现行选中项 = 2
输出调试文本 (列表框1.取项目数值 (列表框1.现行选中项))
```

程序执行后改变“列表框1”的列表项目为a, bb, ccc; 改变项目数值为1, 2, 3; 设置现行选择项为2, 既选择第3个项目; 在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\列表框\列表项目.e”

## 12) “数据源”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

```
列表框1.数据源 = “数据源1”
列表框1.数据列 = “select_”
```

程序执行后将“列表框1”的数据来源和“数据源1”绑定；将列表框显示内容与数据源当前记录中“select\_”列绑定。

## 13) “数据列”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```
列表框1.数据源 = “数据源1”
列表框1.数据列 = “select_”
```

程序执行后将“列表框1”的数据来源和“数据源1”绑定；将列表框显示内容与数据源当前记录中“select\_”列绑定。

## 4.4.2.2 列表框事件

### 1) “列表项被选择”事件

事件原型为：

无返回值，无参数

如果当前所选择的列表项或区域被改变即产生此事件。

例如：

子程序名	返回值类型	公开	备注
_列表框1_列表项被选择			

输出调试文本 (“列表项被选择”)





程序执行后，当“列表框1”中的列表项被选择，在输出面板中显示一段文字。

## 2) “双击选择”事件

事件原型为：

无返回值，无参数

当使用鼠标左键在某列表项目上双击时产生此事件。

例如：

子程序名	返回值类型	公开	备注
列表框1_双击选择			

» 输出调试文本（“双击选择”）

程序运行后，当双击“列表框1”的列表项时在输出面板中显示“双击选择”。

## 4.4.2.3 列表框命令

### 1) “取顶端可见项目”命令

命令原型为：

<整数型> 对象. 取顶端可见项目 ()

本命令用于获取列表框中当前最顶端可见项目的索引。0 为项目一，1 为项目二，以此类推。失败返回 -1。

例如：

输出调试文本(列表框\_可见项目.取顶端可见项目())

程序执行后在输出面板中显示“列表框\_可见项目”中当前可见的第一个项目的索引。

### 2) “置顶端可见项目”命令

命令原型为：

<逻辑型> 对象. 置顶端可见项目 (整数型 项目索引)

本命令用于设置列表框中当前最顶端的可见项目，必要时将自动滚动列表框。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置的项索引，0 为项目一，1 为项目二，以此类推。

例如：

列表框\_可见项目.置顶端可见项目(2)

程序执行后设置“列表框\_可见项目”的第3个项目为当前最顶端显示的项目。

### 3) “取项目数”命令

命令原型为：

<整数型> 对象. 取项目数 ()

本命令用于获取列表框项目数量。





例如：

输出调试文本 (列表框\_取项目数.取项目数 (0))

程序执行后在输出面板中显示“列表框\_取项目数”中的项目数量。

#### 4) “取项目数值”命令

命令原型为：

<整数型> 对象. 取项目数值 (整数型 项目索引)

本命令用于获取与指定项目相关联的数值。如果指定项目不存在，将返回 -1。

参数<1>指定了欲获取的项目索引。0 为项目一，1 为项目二，以此类推。

例如：

列表框1.置项目数值 (1, 321)

» 输出调试文本 (列表框1.取项目数值 (1))

程序执行后设置“列表框1”中的第二个项目所关联的数值为“321”；在输出面板中显示第二个项目所关联的数值。

#### 5) “置项目数值”命令

命令原型为：

<逻辑型> 对象. 置项目数值 (整数型 项目索引, 整数型 欲置入的项目数值)

本命令用于设置与指定项目相关联的数值。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置相关数值的项目索引。0 为项目一，1 为项目二，以此类推。

参数<2>指定了与项目相关联的数值。

例如：

列表框1.置项目数值 (1, 321)

» 输出调试文本 (列表框1.取项目数值 (1))

程序执行后设置“列表框1”中的第二个项目所关联的数值为“321”；在输出面板中显示“列表框1”中第二个项目所关联的数值。

#### 6) “取项目文本”命令

命令原型为：

<文本型> 对象. 取项目文本 (整数型 项目索引)

本命令用于获取指定项目的文本。如果指定项目不存在，将返回空文本。

参数<1>指定了欲获取文本的项目索引。0 为项目一，1 为项目二，以此类推。

例如：

列表框1.置项目文本 (0, “易语言”)

» 输出调试文本 (列表框1.取项目文本 (0))

程序执行后设置“列表框1”中第一个项目的文本为“易语言”；在输出面板中显示“列表框1”中第一个项目的文本。





## 7) “置项目文本”命令

命令原型为:

&lt;逻辑型&gt; 对象. 置项目文本 (整数型 项目索引, 文本型 欲置入的项目文本)

本命令用于设置指定项目的文本。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定了欲设置新文本项目的索引。0 为项目一，1 为项目二，以此类推。

参数&lt;2&gt;指定了项目的新文本。

例如:

列表框1. 置项目文本 (0, “易语言”)

» 输出调试文本 (列表框1. 取项目文本 (0))

程序执行后设置“列表框1”中第一个项目的文本为“易语言”；在输出面板中显示“列表框1”中第一个项目的文本。

## 8) “取已选择项目数”命令

命令原型为:

&lt;整数型&gt; 对象. 取已选择项目数 ()

本命令用于获取已被选择项目的数目。

例如:

输出调试文本 (列表框\_选择. 取已选择项目数 ())

程序执行后在输出面板中显示“列表框\_选择”中被选择项目的数量。

## 9) “取所有被选择项目”命令

命令原型为:

&lt;整数型数组&gt; 对象. 取所有被选择项目 ()

本命令用于获取一个整数数组，内含所有当前被选择项目的位置索引。如果当前没有被选择项目，返回空数组。

例如:

变量名	类型	静态	数组	备注
被选索引组	整数型		0	
计次变量	整数型			

被选索引组 = 列表框1. 取所有被选择项目 ()

--- 计次循环首 (取数组成员数 (被选索引组), 计次变量)  
» 输出调试文本 (被选索引组 [计次变量])  
--- 计次循环尾 ()

程序执行后在输出面板中显示“列表框1”中所有被选择项目的索引。

## 10) “是否被选择”命令

命令原型为:





<逻辑型> 对象. 是否被选择 (整数型 项目索引)

本命令用于判断列表框中指定项当前是否被选择。如果指定项目被选择，则返回“真”，否则返回“假”。

参数<1>指定了欲判断项目的索引。0为项目一，1为项目二，以此类推。

例如：

输出调试文本 (列表框\_选择.是否被选择 (0))

程序执行后在输出面板中显示“列表框\_选择”中的第一个项目是否被选择。如果是，显示“真”，否则显示“假”。

#### 11) “选择项目”命令

命令原型为：

<逻辑型> 对象. 选择项目 (整数型 欲选择或取消选择的项目索引, [逻辑型 欲置入的项目选择状态])

本命令用于选择或取消选择列表框中指定项目。成功返回“真”，失败返回“假”。

参数<1>指定了欲操作项目的索引。0为项目一，1为项目二，以此类推。

参数<2>指定了欲操作项目新的状态。如果省略本参数或者参数值为真，则选择该项目，否则取消对该项目的选择。

例如：

列表框\_选择.选择项目 (0, 真)

程序执行后选择“列表框\_选择”中的第一个项目。

#### 12) “取焦点项目”命令

命令原型为：

<整数型> 对象. 取焦点项目 ()

本命令仅在多选列表框中使用，用作获取当前焦点项目的位置索引。如果在单选列表框中使用本命令，将返回当前被选择项目的位置索引。

例如：

列表框1.置焦点项目 (2)

» 输出调试文本 (列表框1.取焦点项目 (0))

程序执行后将焦点移至“列表框1”的第三个项目上；在输出面板中显示当前焦点所在项目的索引。

#### 13) “置焦点项目”命令

命令原型为：

<逻辑型> 对象. 置焦点项目 (整数型 项目索引)

本命令仅在多选列表框中使用，用作设置当前焦点项目。如果在单选列表框中使用本





命令，将设置当前被选择项目。成功返回“真”，失败返回“假”。

参数<1>指定了欲操作的项目索引。0为项目一，1为项目二，以此类推。

例如：

```
列表框1.置焦点项目 (2)  
输出调试文本 (列表框1.取焦点项目 (0))
```

程序执行后将焦点移至“列表框1”的第三个项目上；在输出面板中显示当前焦点所在项目的索引。

#### 14) “加入项目”命令

命令原型为：

<整数型> 对象. 加入项目 (文本型 欲加入项目的文本, [整数型 与欲加入项目相关的数值] )

本命令用于加入指定项目到列表框的尾部，成功返回加入后该项目所处的位置，失败返回 -1。

参数<1>指定了新项目的文本。

参数<2>指定了与新项目相关联的数值。可以被省略。如果省略本参数，默认值为0。

例如：

```
列表框1.清空 (0)  
列表框1.加入项目 (“易语言”, 123)  
列表框1.插入项目 (0, “abc”, 321)  
列表框1.选择 (“易语言”)
```

程序执行后将“列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

#### 15) “插入项目”命令

命令原型为：

<整数型> 对象. 插入项目 (整数型 欲插入的位置, 文本型 欲插入项目的文本, [整数型 与欲插入项目相关的数值] )

本命令用于插入指定项目到列表框的指定位置处，成功返回插入后该项目所处的位置，失败返回 -1。

参数<1>指定了新项目插入的位置，新项目将插入到指定项目前。0 为项目位置一，1 为项目位置二，以此类推。

参数<2>指定了新项目的文本。

参数<3>指定了与新项目关联的数值。可以被省略。如果省略本参数，默认值为0。





例如：

```
列表框1.清空 ()  
列表框1.加入项目 (“易语言”, 123)  
列表框1.插入项目 (0, “abc”, 321)  
列表框1.选择 (“易语言”)
```

程序执行后将“列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

#### 16) “删除项目”命令

命令原型为：

<逻辑型> 对象.删除项目 (整数型 项目索引)

本命令用于删除列表框指定位置处的项目。成功返回“真”，失败返回“假”。

参数<1>指定了欲删除项目的索引。0为项目位置一，1为项目位置二，以此类推。

例如：

```
列表框_项目.删除项目 (0)
```

程序执行后删除“列表框\_项目”中的第一个项目。

#### 17) “清空”命令

命令原型为：

<无返回值> 对象.清空 ()

本命令用于删除列表框中的所有项目。

例如：

```
列表框1.清空 ()  
列表框1.加入项目 (“易语言”, 123)  
列表框1.插入项目 (0, “abc”, 321)  
列表框1.选择 (“易语言”)
```

程序执行后将“列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

#### 18) “选择”命令

命令原型为：

<整数型> 对象.选择 (文本型 欲选择的项目文本)

本命令用于在所有项目中寻找首部包含指定文本的项目。如找到，则选中它，并返回该项目的位置索引，否则返回-1。

参数<1>指定了欲选择的项目文本。





例如：

```
列表框1.清空 0
列表框1.加入项目 (“易语言”, 123)
列表框1.插入项目 (0, “abc”, 321)
列表框1.选择 (“易语言”)
```

程序执行后将“列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

### 4.4.3 选择列表框

#### 4.4.3.1 选择列表框属性

##### 1) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置选择列表框边框的样式。样式值可以是：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
选择列表框_边框.边框 = 5
```

程序执行后将“选择列表框\_边框”的边框改为单线边框式。

##### 2) “自动排序”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否开启选择列表框项目间的自动排序功能。“真”为开启，“假”为关闭。排序方式为按项目文本依次比对ASCII码升序排列。

例如：

```
选择列表框_排序.自动排序 = 真
```

程序执行后打开“选择列表框\_排序”的自动排序功能。

##### 3) “多列”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定选择列表框在项目较多时是否以多列来显示。“真”为使用多列，“假”为不使用。

例如：

```
选择列表框_多列.多列 = 真
```

程序执行后开启“选择列表框\_多列”的多列显示功能。

##### 4) “行间距”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定各列表项目行之间的间距尺寸。单位为像素。

例如：

```
选择列表框_行间距.行间距 = 10
```

程序执行后将“选择列表框\_行间距”的项目之间的距离设为10像素。

#### 5) “单选”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否只允许选择一项列表项。

例如：

```
选择列表框_单选.单选 = 真
```

程序执行后设置“选择列表框\_单选”为单选样式。

#### 6) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择列表框内文字所使用的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```
变量1.字体名称 = “隶书”
```

```
变量1.字体大小 = 30
```

```
选择列表框1.字体 = 变量1
```

程序执行后会为“选择列表框1”中的文字设置一个新的字体。

#### 7) “现行选中项”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性指定选择列表框中现行被选中列表项目的位置，位置值从0开始，-1表示现行没有被选中的列表项。

例如：

```
输出调试文本(选择列表框1.现行选中项)
```

程序执行后，在输出面板中输出“选择列表框1”中当前被选中项目的位置索引。

#### 8) “列表项目”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定各列表项目。

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-28所示。



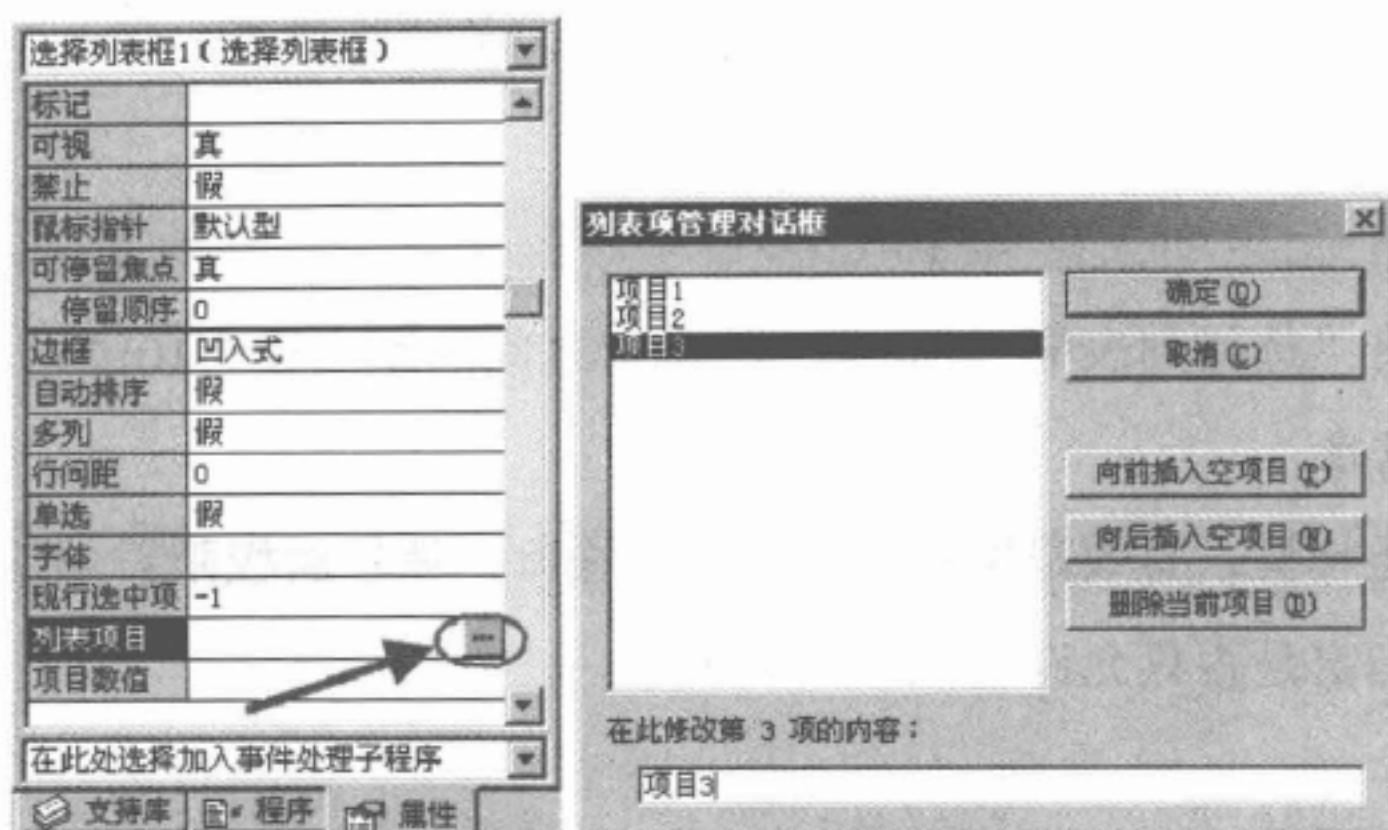


图4-28 打开项目编辑器

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。  
本属性数据存储格式见表4-5。

表4-5 数据存储格式

数据类型 (长度)	说明	数据类型 (长度)	说明
短整数型	项目数	整数型	第二个项目文字长度 (字节数)
整数型	第一个项目文字长度 (字节数)	字节型	第二个项目文字的ASCII码数组
字节型	第一个项目文字的ASCII码数组	.....	.....

例如：

```
选择列表框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98,  
98, 3, 0, 0, 0, 99, 99, 99 }  
选择列表框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }  
选择列表框1.现行选中项 = 2  
» 输出调试文本 (选择列表框1.取项目数值 (选择列表框1.现行选中项))
```

程序执行后改变“选择列表框1”的列表项目为a, bb, ccc; 改变项目数值为1, 2, 3; 设置现行选择项为2, 既选择第3个项目; 在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\选择列表框\列表项目.e”

#### 9) “项目数值”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定与各列表项目相关联的数值，该数值在程序中可以取出或修改。本属性数据格式为：

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-29所示。

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。其数据类型见表4-6。





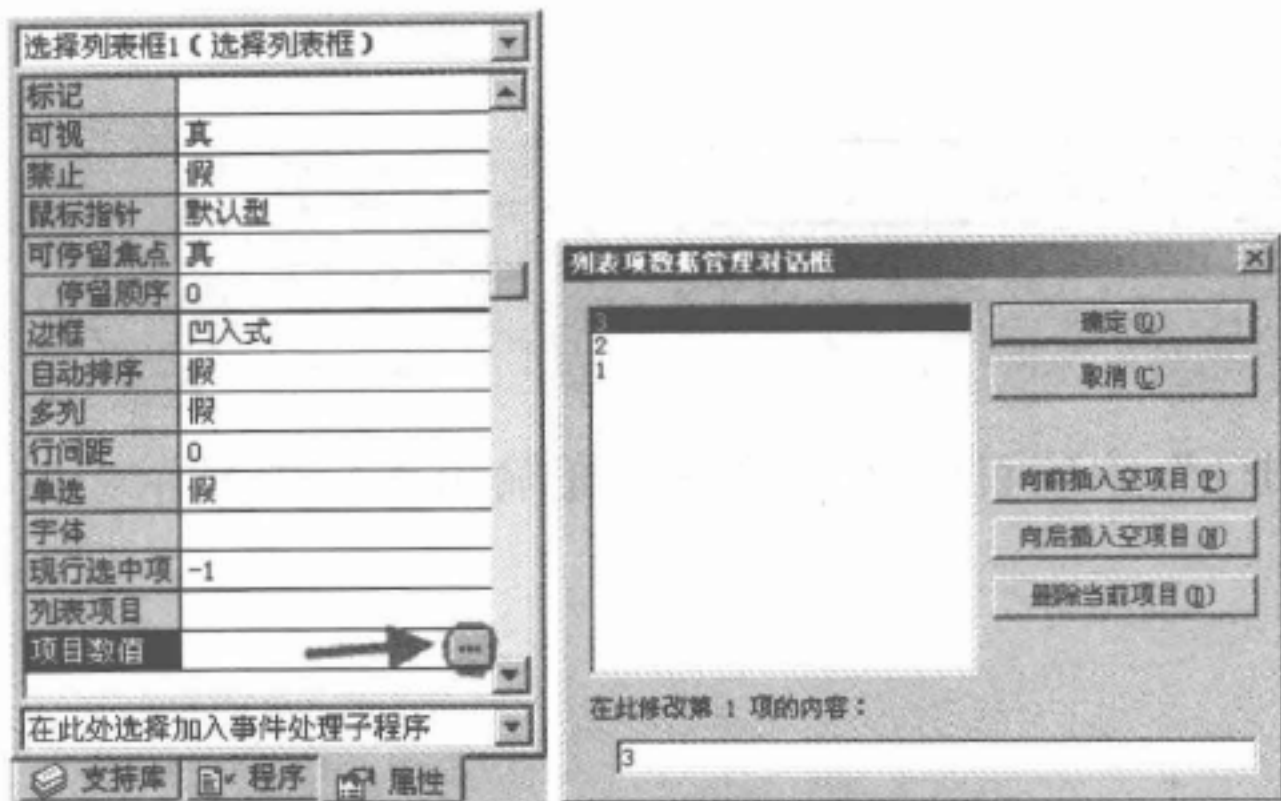


图4-29 打开项目数值编辑器

表4-6 数据存储格式

数据类型 (长度)	说明
整数型	项目一对应的数值
整数型	项目二对应的数值
.....	.....

例如：

```
选择列表框1.列表项目 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98, 3, 0, 0, 0, 99, 99, 99 }
选择列表框1.项目数值 = { 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0 }
选择列表框1.现行选中项 = 2
输出调试文本 (选择列表框1.取项目数值 (选择列表框1.现行选中项))
```

程序执行后改变“选择列表框1”的列表项目为a, bb, ccc；改变项目数值为1, 2, 3；设置现行选择项为2，既选择第3个项目；在输出面板中显示当前选择项的项目数值。

本例程源码见随书光盘中“\图书例程\第四章\选择列表框\列表项目.e”

4.4.3.2 选择列表框事件

1) “列表项被选择”事件

事件原型为：

无返回值；无参数

如果当前所选择的列表项被选择即产生此事件。

例如：

子程序名	返回值类型	公开	备注
选择列表框1_列表项被选择			

```
输出调试文本 (“列表项被选择”)
```

程序执行后，当选择“选择列表框1”中的列表项时在输出面板中显示一段文字。

2) “选中状态被改变”事件

事件原型为：

无返回值；无参数

当选择列表框中的项目选中状态被改变时触发本事件。





例如：

子程序名	返回值类型	公开	备注
_选择列表框1_选中状态被改变			

» 输出调试文本（“选中状态被改变”）

程序执行后，当改变“选择列表框1”中项目的选中状态时在输出面板中显示一段文字。

### 3) “双击选择”事件

事件原型为：

无返回值；无参数

当使用鼠标左键在某列表项目上双击时产生此事件。

例如：

子程序名	返回值类型	公开	备注
_选择列表框1_双击选择			

» 输出调试文本（“双击选择”）

程序运行后，当双击“选择列表框1”的列表项时在输出面板中显示“双击选择”。

## 4.4.3.3 选择列表框命令

### 1) “取顶端可见项目”命令

命令原型为：

<整数型> 对象. 取顶端可见项目 ()

本命令用于获取选择列表框中当前最顶端可见项目的索引。0 为项目一，1 为项目二，以此类推。失败返回 -1。

例如：

选择列表框1. 置顶端可见项目 (2)

» 输出调试文本 (选择列表框1. 取顶端可见项目 () )

程序执行后设置“选择列表框1”中第三个项目为最顶端可见项目；在输出面板中显示“选择列表框1”中当前可见的第一个项索引。

### 2) “置顶端可见项目”命令

命令原型为：

<逻辑型> 对象. 置顶端可见项目 (整数型 项目索引)

本命令用于设置选择列表框中当前最顶端的可见项目，必要时将自动滚动选择列表框。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置的项索引，0 为项目一，1 为项目二，以此类推。





例如：

选择列表框1.置顶端可见项目 (2)

» 输出调试文本 (选择列表框1.取顶端可见项目 (0))

程序执行后设置“选择列表框1”中第三个项目为最顶端可见项目；在输出面板中显示“选择列表框1”中当前可见的第一个项索引。

### 3) “取项目数”命令

命令原型为：

<整数型> 对象.取项目数 ()

本命令用于获取选择列表框项目数量。

例如：

输出调试文本 (选择列表框\_项目数.取项目数 ())

程序执行后在输出面板中显示“选择列表框\_项目数”中的项目数量。

### 4) “取项目数值”命令

命令原型为：

<整数型> 对象.取项目数值 (整数型 项目索引)

本命令用于获取与指定项目相关联的数值。如果指定项目不存在，将返回 -1。

参数<1>指定了欲获取的项目索引。0 为项目一，1 为项目二，以此类推。

例如：

选择列表框1.置项目数值 (1, 321)

» 输出调试文本 (选择列表框1.取项目数值 (1))

程序执行后设置“选择列表框1”中的第二个项目所关联的数值为“321”；在输出面板中显示第二个项目所关联的数值。

### 5) “置项目数值”命令

命令原型为：

<逻辑型> 对象.置项目数值 (整数型 项目索引, 整数型 欲置入的项目数值)

本命令用于设置与指定项目相关联的数值。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置相关数值的项目索引。0 为项目一，1 为项目二，以此类推。

参数<2>指定了与项目相关联的数值。

例如：

选择列表框1.置项目数值 (1, 321)

» 输出调试文本 (选择列表框1.取项目数值 (1))

程序执行后设置“选择列表框1”中的第二个项目所关联的数值为“321”；在输出面板中显示“选择列表框1”中第二个项目所关联的数值。





## 6) “取项目文本”命令

命令原型为:

&lt;文本型&gt; 对象. 取项目文本 (整数型 项目索引)

本命令用于获取指定项目的文本。如果指定项目不存在，将返回空文本。

参数&lt;1&gt;指定了欲获取文本的项目索引。0 为项目一，1 为项目二，以此类推。

例如:

选择列表框1. 置项目文本 (0, “易语言”)

» 输出调试文本 (选择列表框1. 取项目文本 (0))

程序执行后设置“选择列表框1”中第一个项目的文本为“易语言”；在输出面板中显示“选择列表框1”中第一个项目的文本。

## 7) “置项目文本”命令

命令原型为:

&lt;逻辑型&gt; 对象. 置项目文本 (整数型 项目索引, 文本型 欲置入的项目文本)

本命令用于设置指定项目的文本。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定了欲设置新文本项目的索引。0 为项目一，1 为项目二，以此类推。

参数&lt;2&gt;指定了项目的新文本。

例如:

选择列表框1. 置项目文本 (0, “易语言”)

» 输出调试文本 (选择列表框1. 取项目文本 (0))

程序执行后设置“选择列表框1”中第一个项目的文本为“易语言”；在输出面板中显示“选择列表框1”中第一个项目的文本。

## 8) “加入项目”命令

命令原型为:

&lt;整数型&gt; 对象. 加入项目 (文本型 欲加入项目的文本, [整数型 与欲加入项目相关的数值])

本命令用于加入指定项目到选择列表框的尾部，成功返回加入后该项目所处的位置，失败返回 -1。

参数&lt;1&gt;指定了新项目的文本。

参数&lt;2&gt;指定了与新项目相关联的数值。可以被省略。如果省略本参数，默认值为 0。

例如:

选择列表框1. 清空 (0)

选择列表框1. 加入项目 (“易语言”, 123)

选择列表框1. 插入项目 (0, “abc”, 321)

选择列表框1. 选择 (“易语言”)





程序执行后将“选择列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

#### 9) “插入项目”命令

命令原型为：

<整数型> 对象. 插入项目 (整数型 欲插入的位置, 文本型 欲插入项目的文本, [整数型 与欲插入项目相关的数值])

本命令用于插入指定项目到选择列表框的指定位置处，成功返回插入后该项目所处的位置，失败返回 -1。

参数<1>指定了新项目插入的位置，新项目将插入到指定项目前。0 为项目位置一，1 为项目位置二，以次类推。

参数<2>指定了新项目的文本。

参数<3>指定了与新项目关联的数值。可以被省略。如果省略本参数，默认值为 0。

例如：

```
选择列表框1. 清空 ()  
选择列表框1. 加入项目 ("易语言", 123)  
选择列表框1. 插入项目 (0, "abc", 321)  
选择列表框1. 选择 ("易语言")
```

程序执行后将“选择列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

#### 10) “删除项目”命令

命令原型为：

<逻辑型> 对象. 删除项目 (整数型 项目索引)

本命令用于删除选择列表框指定位置处的项目。成功返回“真”，失败返回“假”。

参数<1>指定了欲删除项目的索引。0 为项目位置一，1 为项目位置二，以此类推。

例如：

```
选择列表框_删除. 删除项目 (0)
```

程序执行后删除“选择列表框\_删除”中的第一个项目。

#### 11) “清空”命令

命令原型为：

<无返回值> 对象. 清空 ()

本命令用于删除选择列表框中的所有项目。





例如：

```
选择列表框1.清空 0
选择列表框1.加入项目 (“易语言”, 123)
选择列表框1.插入项目 (0, “abc”, 321)
选择列表框1.选择 (“易语言”)
```

程序执行后将“选择列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

## 12) “选择”命令

命令原型为：

<整数型> 对象. 选择 (文本型 欲选择的项目文本)

本命令用于在所有项目中寻找首部包含指定文本的项目，如找到，则选中它，并返回该项目的位置索引，否则返回 -1。

参数<1>指定了欲选择的项目文本。

例如：

```
选择列表框1.清空 0
选择列表框1.加入项目 (“易语言”, 123)
选择列表框1.插入项目 (0, “abc”, 321)
选择列表框1.选择 (“易语言”)
```

程序执行后将“选择列表框1”中原有的项目全部删除；加入文本为“易语言”，关联数值为“123”的新项目；在第一个项目前插入文本为“abc”，关联数值为“321”的新项目；选择项目文本为“易语言”的项目。

## 13) “是否被选中”命令

命令原型为：

<逻辑型> 对象. 是否被选中 (整数型 项目索引)

本命令用于判断指定项目对应的选择框是否被选中，选中则返回“真”，否则返回“假”。

参数<1>指定了欲判断项目的索引。0 为项目位置一，1 为项目位置二，以此类推。

例如：

```
选择列表框1.选中项目 (0, 真)
» 输出调试文本 (选择列表框1.是否被选中 (0))
```

程序执行后将“选择列表框1”的第一个项目选中；在输出面板中显示“选择列表框1”中的第一个项目是否被选中。如果是，显示“真”，否则显示“假”。



## 14) “选中项目”命令

命令原型为:

<逻辑型> 对象. 选中项目 (整数型 欲选择或取消选择的项目索引, [逻辑型 欲置入的项目选择状态])

本命令用于选中或取消选中与指定项目对应的选择框。成功返回“真”，失败返回“假”。

参数<1>指定了欲操作项目的索引。0 为项目一，1 为项目二，以此类推。

参数<2>指定了欲操作项目新的状态。如果省略本参数或者参数值为“真”，则选中该项目，否则取消对该项目的选中。

例如:

选择列表框1.选中项目 (0, 真)

» 输出调试文本 (选择列表框1.是否被选中 (0))

程序执行后将“选择列表框1”的第一个项目选中；在输出面板中显示“选择列表框1”中的第一个项目是否被选中。如果是，显示“真”，否则显示“假”。

## 15) “是否被允许”命令

命令原型为:

<逻辑型> 对象. 是否被允许 (整数型 项目索引)

本命令用于判断指定项目对应的选择框是否允许操作，允许则返回“真”，否则返回“假”。

参数<1>指定了欲判断项目的索引。0 为项目一，1 为项目二，以此类推。

例如:

选择列表框1.允许 (0, 假)

» 输出调试文本 (选择列表框1.是否被允许 (0))

程序执行后禁止“选择列表框1”中第一个项目；在输出面板中显示“选择列表框1”中的第一个项目是否被允许。如果是，显示“真”，否则显示“假”。

## 16) “允许”命令

命令原型为:

<逻辑型> 对象. 允许 (整数型 欲允许或禁止操作的项目索引, [逻辑型 允许或禁止])

本命令用于允许或禁止对指定项目进行选择操作。成功返回“真”，失败返回“假”。

参数<1>指定了欲操作项目的索引。0 为项目一，1 为项目二，以此类推。





参数<2>指定了欲操作项目新的状态。如果省略本参数或者参数值为“真”，则允许该项目，否则禁止对该项目进行选中操作。

例如：

```
选择列表框1.允许 (0, 假)
» 输出调试文本 (选择列表框1.是否被允许 (0))
```

程序执行后禁止“选择列表框1”中第一个项目；在输出面板中显示“选择列表框1”中的第一个项目是否被允许。如果是，显示“真”，否则显示“假”。

## 4.5 系统类组件

### 4.5.1 通用对话框

#### 4.5.1.1 通用对话框属性

##### 1) “类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置通用对话框所对应的功能。可供选择的属性值有：0（打开文件）；1（保存文件）；2（字体选择）；3（打开帮助）。

例如：

```
通用对话框_类型.类型 = 1
```

程序执行后将“通用对话框\_类型”的功能类型设为保存文件样式。

##### 2) “标题”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定通用对话框的标题。本属性仅适用于打开文件和保存文件对话框。若本属性为空，“类型”属性为0时，默认标题为“打开”；若本属性为空，“类型”属性为1时，默认标题为“另存为”。

例如：

```
通用对话框_标题.标题 = “打开文件”
```

程序执行后将“通用对话框\_标题”的标题设为“打开文件”。

##### 3) “文件名”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
从本属性中读取的文件名包括驱动器符和全路径目录，同时本属性也用作初始对话框中的文件名编辑框。本属性仅适用于打开文件和保存文件对话框。



例如：

```
通用对话框1.类型 = 0
... 如果真 (通用对话框1.打开 () = 真)
    输出调试文本 (通用对话框1.文件名)
```

程序执行后，如果在通用对话框中按下“打开”按钮，则在输出面板中显示被选中的文件全路径名。

#### 4) “过滤器”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置在打开文件或保存文件时在通用对话框中显示的文件条件。过滤器文本由单个或多个成对的文本串组成，每对文本串的第一个描述显示形式，如：“文本文件 (\*.txt)”；第二个指定实际的过滤匹配符，如：“\*.txt”，所有各文本串之间用“|”隔开。

例如：

```
通用对话框1.过滤器 = “文本文件 (*.txt) | *.txt | 所有文件 (*.*) | *.*”
通用对话框1.初始过滤器 = 1
```

程序执行后设置两组过滤器并默认使用第二组过滤器。

#### 5) “初始过滤器”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性仅适用于打开文件和保存文件对话框。如果“过滤器”属性有效且不为空，则本属性用于指定默认使用的过滤器序号。过滤器序号从0开始，0表示默认使用第一个过滤器。

例如：

```
通用对话框1.过滤器 = “文本文件 (*.txt) | *.txt | 所有文件 (*.*) | *.*”
通用对话框1.初始过滤器 = 1
```

程序执行后设置两组过滤器并默认使用第二组过滤器。

#### 6) “初始目录”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定当打开对话框时所自动跳转到的目录。本属性仅适用于打开文件和保存文件对话框。

例如：

```
通用对话框_目录.初始目录 = “C:\”
```

程序执行后设置“通用对话框\_目录”的初始目录为“C:\”。





## 7) “默认文件后缀” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当用户没有输入文件的后缀名称时所自动使用的文件后缀名称。

例如：

通用对话框\_后缀.默认文件后缀 = “.e”

程序执行后设置“通用对话框\_后缀”的默认后缀为“.e”。

## 8) “创建时提示” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于当“类型”为打开文件对话框时，如果用户指定了一个不存在的文件名称，是否提示用户即将创建它。“真”为自动创建，“假”为不创建。

例如：

通用对话框\_提示.创建时提示 = 真

程序执行后启用“通用对话框\_提示”的文件不存在则自动创建功能。

## 9) “文件必须存在” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于当类型为打开文件对话框时，是否允许用户指定一个不存在的文件。  
“真”为必须存在，“假”为可以不存在。

例如：

通用对话框\_存在.文件必须存在 = 真

程序执行后设置“通用对话框\_存在”在打开文件时，欲打开的文件必须存在。

## 10) “文件覆盖提示” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于当“类型”为保存文件对话框时，如果用户指定了一个已经存在的文件，是否询问用户确定覆盖此文件。“真”为提示，“假”为不提示。

例如：

通用对话框\_覆盖.文件覆盖提示 = 真

程序执行后设置“通用对话框\_覆盖”开启文件覆盖时提示功能。

## 11) “目录必须存在” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否允许用户指定一个不存在的目录。“真”为不允许指定不存在的目录，“假”为允许指定不存在的目录。本属性仅适用于打开文件和保存文件对话框。

例如：

通用对话框\_目录.目录必须存在 = 真



程序执行后设置“通用对话框\_目录”在打开和保存文件时不允许指定到一个不存在的目录。

#### 12) “不改变目录”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在对话框关闭后是否自动返回到进入对话框时的文件目录。“真”为自动返回，“假”为下次打开时仍然在上次关闭时的目录。本属性仅适用于打开文件和保存文件对话框。

例如：

```
通用对话框_目录.不改变目录 = 真
```

程序执行后设置“通用对话框\_目录”关闭后自动复原到初始目录。

#### 13) “字体颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框中颜色选择时的默认颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色  
通用对话框1.加粗 = 真  
通用对话框1.倾斜 = 真  
通用对话框1.删除线 = 假  
通用对话框1.下划线 = 真  
通用对话框1.字体名称 = “黑体”  
通用对话框1.字体大小 = 20  
通用对话框1.打开()
```

程序执行后设置“通用对话框1”的默认字体为绿色；启用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 14) “加粗”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框中加粗效果的初始状态。“真”为使用，“假”为不使用。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色  
通用对话框1.加粗 = 真  
通用对话框1.倾斜 = 真  
通用对话框1.删除线 = 假
```





```
通用对话框1.下划线 = 真
通用对话框1.字体名称 = “黑体”
通用对话框1.字体大小 = 20
通用对话框1.打开 ()
```

程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 15) “倾斜”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框中倾斜效果的初始状态。“真”为使用，“假”为不使用。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色
通用对话框1.加粗 = 真
通用对话框1.倾斜 = 真
通用对话框1.删除线 = 假
通用对话框1.下划线 = 真
通用对话框1.字体名称 = “黑体”
通用对话框1.字体大小 = 20
通用对话框1.打开 ()
```

程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 16) “删除线”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框中是否开启删除线效果。“真”为开启，“假”为不开启。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色
通用对话框1.加粗 = 真
通用对话框1.倾斜 = 真
通用对话框1.删除线 = 假
通用对话框1.下划线 = 真
通用对话框1.字体名称 = “黑体”
通用对话框1.字体大小 = 20
通用对话框1.打开 ()
```





程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 17) “下划线”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框中是否开启下划线效果。“真”为开启，“假”为不开启。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色
通用对话框1.加粗 = 真
通用对话框1.倾斜 = 真
通用对话框1.删除线 = 假
通用对话框1.下划线 = 真
通用对话框1.字体名称 = “黑体”
通用对话框1.字体大小 = 20
通用对话框1.打开()
```

程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 18) “字体名称”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通用对话框的初始字体名称。如果不指定字体名称和字体大小，将自动使用系统默认字体。如果指定了字体大小但没有指定字体名称，将自动使用“宋体”。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色
通用对话框1.加粗 = 真
通用对话框1.倾斜 = 真
通用对话框1.删除线 = 假
通用对话框1.下划线 = 真
通用对话框1.字体名称 = “黑体”
通用对话框1.字体大小 = 20
通用对话框1.打开()
```

程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 19) “字体大小”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定通用对话框的初始字体大小，单位为1/72英寸。如果不指定字体名称和字体大小，将自动使用系统默认字体。如果指定了字体大小但没有指定字体名称，将自动使用“宋体”。本属性仅适用于字体类型对话框。

例如：

```
通用对话框1.字体颜色 = #绿色  
通用对话框1.加粗 = 真  
通用对话框1.倾斜 = 真  
通用对话框1.删除线 = 假  
通用对话框1.下划线 = 真  
通用对话框1.字体名称 = “黑体”  
通用对话框1.字体大小 = 20  
通用对话框1.打开()
```

程序执行后设置“通用对话框1”的默认字体为绿色；起用加粗和斜体效果；不开启删除线样式；开启下划线样式；设置默认字体为黑体；字体大小为20；显示“通用对话框1”。

#### 20) “帮助文件名”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定欲打开帮助文件的名称，如果没有指定帮助文件，将使用默认帮助文件名称，即本易程序的文件名称（不含其后缀）+“.hlp”后缀。本属性仅适用于打开帮助类型对话框。

例如：

```
通用对话框1.帮助文件名 = “C:\WINDOWS\Help\calc.hlp”  
通用对话框1.帮助命令 = 2  
通用对话框1.帮助标志值 = 123  
通用对话框1.打开()
```

程序执行后设置“通用对话框1”的欲打开的帮助文件名为

“C:\WINDOWS\Help\calc.hlp”（假设文件存在）；设置打开方式为弹出指定标志的帮助；设置欲弹出的帮助标志值为“123”；显示“通用对话框1”即显示所指定的帮助内容。

#### 21) “帮助命令”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打开帮助的功能类型。可供选择的属性值有：0（显示帮助主题）；1（显示指定标志信息）；2（弹出指定标志信息）；3（确定帮助已显示）；4（显示帮助软件使用方法）；5（退出帮助）。

本属性仅适用于打开帮助类型对话框。





例如：

```
通用对话框1.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
通用对话框1.帮助命令 = 2  
通用对话框1.帮助标志值 = 123  
通用对话框1.打开 ()
```

程序执行后设置“通用对话框1”的欲打开的帮助文件名为

“C:\WINDOWS\Help\calc.hlp”（假设文件存在）；设置打开方式为弹出指定标志的帮助；设置欲弹出的帮助标志值为“123”；显示“通用对话框1”即显示所指定的帮助内容。

## 22) “帮助标志值” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性仅在通用对话框“类型”属性为打开帮助且帮助命令为“显示指定标志信息”或“弹出指定标志信息”时才有效。本属性用于指定相应的帮助标志数值。

例如：

```
通用对话框1.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
通用对话框1.帮助命令 = 2  
通用对话框1.帮助标志值 = 123  
通用对话框1.打开 ()
```

程序执行后设置“通用对话框1”的欲打开的帮助文件名为

“C:\WINDOWS\Help\calc.hlp”（假设文件存在）；设置打开方式为弹出指定标志的帮助；设置欲弹出的帮助标志值为“123”；显示“通用对话框1”即显示所指定的帮助内容。

### 4.5.1.2 通用对话框命令

“打开”命令原型为：

<逻辑型> 对象. 打开 ()

本命令用于打开当前类型的对话框，并返回一个逻辑值。对于类型为“打开文件”、“保存文件”、“字体选择”的对话框，如果为“真”，表示用户已通过该对话框输入了有效数据，否则表示用户取消了该对话框，且没有输入任何有效数据。对于类型为“打开帮助”的对话框，如打开帮助成功则返回“真”，否则返回“假”。

例如：

```
通用对话框1.帮助文件名 = "C:\WINDOWS\Help\calc.hlp"  
通用对话框1.帮助命令 = 2  
通用对话框1.帮助标志值 = 123  
通用对话框1.打开 ()
```

程序执行后设置“通用对话框1”的欲打开的帮助文件名为：





“C:\WINDOWS\Help\calc.hlp”（假设文件存在）；设置打开方式为弹出指定标志的帮助；设置欲弹出的帮助标志值为“123”；显示“通用对话框1”即显示所指定的帮助内容。

## 4.5.2 文件框

### 4.5.2.1 文件框属性

#### 1) “目录”属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取、更改  
本属性用于指定文件框的现行目录。

例如：

```
文件框_目录.目录 = "C:\Windows"
```

程序执行后将“文件框\_目录”的现行目录设为“C:\Windows”。

#### 2) “被选择文件”属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性用于记录所有现行被选择的文件，注意不包含路径。如果有多个文件被选择，各文件名之间用分号“;”隔开。

例如：

子程序名	返回值类型	公开	备注
_文件框1_选择文件被改变			

» 输出调试文本（文件框1.被选择文件）

程序执行后，当“文件框1”中选择的文件被改变后在输出面板中显示当前被选择的文件名。

#### 3) “允许选择多项”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许在文件框中同时选择多个项目。“真”为允许，“假”为不允许。

例如：

```
文件框_选择.允许选择多项 = 真
```

程序执行后，允许“文件框\_选择”同时选择多个项目。

#### 4) “通配符”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定允许进入文件框的文件类型。可以同时指定多个通配符，各通配符之间用分号“;”隔开。通配符：“\*”代表任意多个字符；“?”代表任意单个字符。





例如：

```
文件框_通配符.通配符 = "*.exe;*.b?p"
```

程序执行后设置“文件框\_通配符”中允许显示以“.exe”为结尾和以“.b（任意单字符）p”为结尾的文件。

#### 5) “通常”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许没有设置存档、只读、系统、隐藏文件属性的文件在文件框中显示。“真”为允许，“假”为不允许。

例如：

```
文件框1.通常 = 真  
文件框1.存档 = 真  
文件框1.只读 = 真  
文件框1.系统 = 真  
文件框1.隐藏 = 真
```

程序执行后，允许在“文件框1”中显示拥有“通常、存档、只读、系统、隐藏”，其中任一文件属性的文件。

#### 6) “存档”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许设置了存档文件属性的文件在文件框中显示。“真”为允许，“假”为不允许。

例如：

```
文件框1.通常 = 真  
文件框1.存档 = 真  
文件框1.只读 = 真  
文件框1.系统 = 真  
文件框1.隐藏 = 真
```

程序执行后，允许在“文件框1”中显示拥有“通常、存档、只读、系统、隐藏”，其中任一文件属性的文件。

#### 7) “只读”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许设置了只读文件属性的文件在文件框中显示。“真”为允许，“假”为不允许。

例如：

```
文件框1.通常 = 真  
文件框1.存档 = 真
```



```
文件框1.只读 = 真
文件框1.系统 = 真
文件框1.隐藏 = 真
```

程序执行后，允许在“文件框1”中显示拥有“通常、存档、只读、系统、隐藏”，其中任一文件属性的文件。

#### 8) “系统”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许设置了系统文件属性的文件在文件框中显示。“真”为允许，“假”为不允许。

例如：

```
文件框1.通常 = 真
文件框1.存档 = 真
文件框1.只读 = 真
文件框1.系统 = 真
文件框1.隐藏 = 真
```

程序执行后，允许在“文件框1”中显示拥有“通常、存档、只读、系统、隐藏”，其中任一文件属性的文件。

#### 9) “隐藏”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许设置了隐藏文件属性的文件在文件框中显示。“真”为允许，“假”为不允许。

例如：

```
文件框1.通常 = 真
文件框1.存档 = 真
文件框1.只读 = 真
文件框1.系统 = 真
文件框1.隐藏 = 真
```

程序执行后，允许在“文件框1”中显示拥有“通常、存档、只读、系统、隐藏”，其中任一文件属性的文件。

#### 10) “最小日期”属性

**属性类型：**日期时间型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定允许在文件框中显示的文件的日期。

例如：

```
文件框1.最小日期 = [2000年1月1日]
文件框1.最大日期 = [2010年1月1日]
```



程序执行后设置“文件框1”中显示的文件必须满足修改时间为2000年1月1日到2010年1月1日之间的文件。

#### 11) “最大日期”属性

**属性类型：**日期时间型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定允许在文件框中显示的文件的最大日期。若本属性值为1899年12月30日或更早，则依据文件修改时间过滤功能无效。

例如：

```
文件框1.最小日期 = [2000年1月1日]
文件框1.最大日期 = [2010年1月1日]
```

程序执行后设置“文件框1”中显示的文件必须满足修改时间为2000年1月1日到2010年1月1日之间的文件。

#### 12) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定文件框边框的显示样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
文件框_边框.边框 = 4
```

程序执行后将“文件框\_边框”的边框设为镜框式。

#### 13) “文件颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定文件框中文字的显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
文件框1.文本颜色 = #绿色
文件框1.背景颜色 = #黑色
```

程序执行后将“文件框1”中的文字显示颜色设为绿色；文字的背景颜色设为黑色。

#### 14) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定文件框中文字的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
文件框1.文本颜色 = #绿色
文件框1.背景颜色 = #黑色
```





程序执行后将“文件框1”中的文字显示颜色设为绿色；文字的背景颜色设为黑色。

#### 15) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定文件框中文字的字体。

例如：

变量名	类 型	静态	数组	备 注
变量	字体			

变量.字体名称 = “幼圆”

变量.字体大小 = 20

文件框1.字体 = 变量

程序执行后使“文件框1”中的文字应用新字体。

### 4.5.2.2 文件框事件

#### 1) “选择文件被改变”事件

事件原型为：

**返回值类型：**无返回值，无参数

当文件框中选择的文件被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备 注
_文件框1_选择文件被改变			

» 输出调试文本 (文件框1.被选择文件)

程序执行后，当“文件框1”中选择的文件被改变后在输出面板中显示当前被选择的文件名。

#### 2) “双击选择”事件

事件原型为：

**返回值类型：**无返回值，无参数

当使用鼠标左键在某列表项目上双击时触发本事件。

例如：

子程序名	返回值类型	公开	备 注
_文件框1_双击选择			

» 输出调试文本 (文件框1.被选择文件)

程序执行后，当在“文件框1”中双击选择某文件后在输出面板中显示当前被选择的文件名。





4.5.3 目录框

4.5.3.1 目录框属性

1) “目录” 属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取、更改  
本属性用于指定目录框的现行目录。

例如：

```
目录框_目录.目录 = "C:\Windows"
```

程序执行后将“目录框\_目录”的现行目录设为“C:\Windows”。

2) “边框” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置目录框的边框样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
目录框_边框.边框 = 4
```

程序执行后将“目录框\_边框”的边框样式设为镜框式。

3) “背景颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置目录框内背景的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
目录框_背景.背景颜色 = #黑色
```

程序执行后将“目录框\_背景”的背景颜色设为黑色。

4) “字体” 属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置目录框文字的字体。

例如：

变量名	类型	静态	数组	备注
变量	字体			

变量.字体名称 = “幼圆”  
变量.字体大小 = 20  
目录框1.字体 = 变量

程序执行后改变“目录框1”中文字的字体。

4.5.3.2 目录框事件

事件原型为：





返回值类型：无返回值，无参数

当目录框的现行目录被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备注
_目录框1_目录被改变			

» 输出调试文本（目录框1. 目录）

程序执行后，当“目录框1”的目录被改变后在输出面板中显示“目录框1”的现行目录。

## 4.5.4 驱动器框

### 4.5.4.1 驱动器框属性

#### 1) “驱动器”属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取、更改

本属性用于指定当前驱动器符，如果不指定，默认为程序当前驱动器。驱动器符为大写字母A~Z。

例如：

驱动器框\_驱动器.驱动器 = “C”

程序执行后设置“驱动器框\_驱动器”当前显示驱动器为“C”盘。

#### 2) “类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定可用的驱动器类型。可供选择的属性值有：0（全部驱动器）；1（可抽取式驱动器）；2（固定驱动器）；3（光盘驱动器）；4（网络驱动器）；5（内存虚拟驱动器）。

例如：

驱动器框\_类型.类型 = 2

程序执行后设置“驱动器框\_类型”只显示类型为“硬盘”的盘符。

#### 3) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定驱动器框内文字的颜色。属性值为RGB颜色值，可以通过“取颜色”函数获取。

例如：

驱动器框1.文本颜色 = #绿色  
驱动器框1.背景颜色 = #黑色

程序执行后将“驱动器框1”中的文本颜色设为绿色；背景颜色设为黑色。





## 4) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定驱动器框内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
驱动器框1.文本颜色 = #绿色
驱动器框1.背景颜色 = #黑色
```

程序执行后将“驱动器框1”中的文本颜色设为绿色；背景颜色设为黑色。

## 5) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定驱动器框中文字的字体。

例如：

变量名	类型	静态	数组	备注
变量	字体			

变量.字体名称 = “隶书”

变量.字体大小 = 20

驱动器框1.字体 = 变量

程序执行后为“驱动器框1”中的文字应用新的字体。

## 4.5.4.2 驱动器框事件

“驱动器被改变”事件原型为：

**返回值类型：**无返回值；**无参数**

当驱动器框现行驱动器被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备注
_驱动器框1_驱动器被改变			

» 输出调试文本 (驱动器框1.驱动器)

程序执行后，当“驱动器框1”的现行驱动器被改变后在输出面板中显示改变后的驱动器盘符。

## 4.5.5 端口

## 4.5.5.1 端口属性

## 1) “端口号”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于设置所控制的端口号。端口号从 1 开始。

## 2) “波特率” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通信端口所使用的波特率，比较常用的波特率有：110，300，600，1200，2400，4800，9600，14400，19200，38400，56000，57600，115200，128000，256000。

## 3) “数据位数” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定每次发送的数据位数。

## 4) “停止位数” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定停止标志的位数。可以为以下值之一：1(1位)；2(1.5位)；3(2位)。

## 5) “奇偶校验” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定通信时对数据是否进行奇偶校验。“真”为使用，“假”为不使用。

## 6) “事件字符” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定用作触发事件的字符。只使用文本的首字符，为空表示不支持事件字符。

## 7) “等待时间” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在进行端口读写操作时最长等待时间。单位为毫秒。

## 8) “自动启动” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

在对端口进行操作之前必须首先启动，本属性用于指定当组件被载入时是否自动启动。“真”为自动启动，“假”为不自动启动。

## 9) “奇偶校验方案” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

当使用奇偶校验时，本属性用于指定通信时所使用的具体奇偶校验位方案。可供选择的属性值有：0(无)；1(奇校验)；2(偶校验)；3(标志校验)；4(空白校验)。

## 10) “流控制方案” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

指定通信时所使用的流控制方案。可供选择的属性值有：0(无)；1(CtsRts)；2(CtsDtr)；3(DsrRts)；4(DsrDtr)；5(XonXoff)。





## 11) “Rts流控制” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

指定通信时是否启用Rts流控制。可供选择的属性值有：0（默认）；1（禁止）；2（启用）。

## 12) “Dtr流控制” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

指定通信时是否启用Dtr流控制。可供选择的属性值有：0（默认）；1（禁止）；2（启用）。

## 4.5.5.2 端口事件

## 1) “收到信号” 事件

事件原型为：

返回值类型：无返回值

参数一：参数名：信号类型类型：整数型

当通信端口的信号状态发生改变时，会产生本事件。

参数<1>指示了具体检测到信号状态的类型，为以下值之一：1（BREAK 信号）；2（CTS 上升沿信号）；3（DSR 上升沿信号）；4（ERR 信号）；5（RING 信号）；6（RLSD 上升沿信号）；7（事件字符到达）；8（CTS 下降沿信号）；9（DSR 下降沿信号）；10（RLSD 下降沿信号）。

## 2) “数据到达” 事件

事件原型为：

返回值类型：无返回值

参数一：参数名：数据字节值 类型：整数型

每当通信端口接收到一个数据字节，就会产生本事件。

本命令的第一个参数保存了所接收到的数据字节值。

## 4.5.5.3 端口命令

## 1) “启动” 命令

命令原型为：

<逻辑型> 对象. 启动 ()

本命令用于启动或重新启动对指定端口的操作。在对端口进行操作之前必须首先启动，如在端口启动后又更改了端口属性必须重新启动。成功返回“真”，失败返回“假”。

## 2) “停止” 命令

命令原型为：

<无返回值> 对象. 停止 ()

本命令用于关闭已经启动的指定端口。





## 3) “发送数据”命令

命令原型为:

&lt;逻辑型&gt; 对象. 发送数据 (通用型 欲发送数据)

本命令用于从端口发送指定的数据。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定了欲发送的数据。欲发送数据必须是系统基本数据类型。

## 4) “信号操作”命令

命令原型为:

&lt;逻辑型&gt; 对象. 信号操作 (整数型 操作类型, 整数型 欲操作信号类型)

本命令可以设置或清除通信端口上指定信号的状态。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定了欲做操作的类型。

本参数可以为以下常量值之一：1（#清除信号）；2（#发送或置位）。

参数<2>指定了欲操作的信号类型。本参数可以为以下常量值之一：1（#DTR信号）；2（#RTS信号）；3（#Break信号）。

## 4.6 图形类组件

### 4.6.1 图片框

图片框属性

## 1) “边框”属性

**属性类型：**整数型，**有效范围：**设计时、编程时，**编程时权限：**使用、读取、更改

本属性用于设置图片框边框的样式。样式值可以是：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如:

图片框\_边框.边框 = 5

程序执行后将“图片框\_边框”的边框改为单线边框。

## 2) “背景颜色”属性

**属性类型：**整数型，**有效范围：**设计时、编程时，**编程时权限：**使用、读取、更改

本属性用于设置图片框内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如:

图片框\_背景.背景颜色 = #红色

程序执行后会将“图片框\_背景”的背景颜色设为红色。





### 3) “图片” 属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定显示在图片框上的图片数据。图片数据格式可以是JPG、BMP、GIF、ICO、CUR。

例如：

```
图片框1.图片 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
图片框1.显示方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）显示在图片框中；设置图片显示方式为“缩放”。

### 4) “显示方式” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果设定了图片，本属性用于指定图片框上图片的显示方式。显示方式可以是以下值之一：0（图片居左上）；1（缩放图片）；2（图片居中）。

例如：

```
图片框1.图片 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
图片框1.显示方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）显示在图片框中；设置图片显示方式为“缩放”。

### 5) “播放动画” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

当图片为GIF动画格式时，通过改变此属性可控制其播放或停止。如在设计时置本属性为“真”，运行时动画在图片框被创建后即自动开始播放。

例如：

```
图片框_动画.播放动画 = 真
```

程序执行后，自动开始播放“图片框\_动画”中的GIF动画。

### 6) “数据源” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

```
图片框1.数据源 = “数据源1”  
图片框1.数据列 = “pic”
```

程序执行后将“图片框1”的数据来源和“数据源1”绑定；将图片框显示内容与数据





源当前记录中“pic”列绑定。

#### 7) “数据列”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```
图片框1.数据源 = “数据源1”
图片框1.数据列 = “pic”
```

程序执行后将“图片框1”的数据来源和“数据源1”绑定；将图片框显示内容与数据源当前记录中“pic”列绑定。

## 4.6.2 画板

### 4.6.2.1 画板属性

#### 1) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板边框的样式。样式值可以是：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
画板_边框.边框 = 5
```

程序执行后将“画板\_边框”的边框改为单线边框。

#### 2) “画板背景色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。如果“自动重画”属性为“真”，则不支持透明色。

例如：

```
画板_背景色.画板背景色 = #红色
```

程序执行后会将“画板\_背景色”的背景颜色设为红色。

#### 3) “底图”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定显示在画板背景上的图片。

例如：

```
画板1.底图 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)
画板1.底图方式 = 1
```





程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“画板1”的底图；将底图显示方式设为平铺。

#### 4) “底图方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果设定了底图，本属性用于指定画板背景上底图的显示方式。显示方式有：0（图片居上）；1（图片平铺）；2（图片居中）；3（缩放图片）。

例如：

```
画板1.底图 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
画板1.底图方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“画板1”的底图；将底图显示方式设为平铺。

#### 5) “自动重画”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果本属性设置为“真”，系统将自动把所有在画板上进行的绘画操作的结果保存到内存中，当以后需要重画时，系统简单地把所保存的绘画结果复制到画板上，而不再产生“绘画”事件。开启本功能需要消耗一定数量的内存。另外，如果本属性为“真”且在底图方式为居中时调整画板尺寸，将无法保留以前的绘画数据。

例如：

```
画板_重画.自动重画 = 真
```

程序执行后开启“画板\_重画”的自动重画功能。

#### 6) “绘画单位”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定绘画时所使用的单位，坐标系的 X 轴从左到右，Y 轴从上到下。本属性可以是以下值之一：0（像素点）；1（0.1毫米）；2（0.01毫米）；3（0.01英寸）；4（0.001英寸）；5（1/1440英寸）。

例如：

```
画板_绘画.绘画单位 = 2
```

程序执行后会将“画板\_绘画”的绘画单位设为0.01毫米。

#### 7) “画笔类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在画板绘画时的画笔类型。本属性可以是以下值之一：0（空笔）；1（直线）；2（划线）；3（点线）；4（内划线）；5（双点划线）；6（内直线）。

例如：

```
画板_画笔.画笔类型 = 3
```





程序执行后将“画板\_画笔”的画笔类型设为点线形式。

#### 8) “画出方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定画板画笔的画出方式。有关各种画出方式的说明为：0（黑色：始终画出黑色）；1（非或笔：或笔的相反）；2（与非笔：背景色以及画笔反相二者共有颜色的组合）；3（非复制笔：复制笔的相反）；4（与笔非：画笔以及已有颜色反相二者共有颜色的组合）；5（反转：已有颜色的反相）；6（异或笔：画笔的颜色以及已有颜色的组合，只取其一）；7（非与笔：与笔的相反）；8（与笔：画笔和已有二者共有颜色的组合）；9（非异或笔：异或笔的相反）；10（无操作：原有保持不变。该设置实际上关闭画笔）；11（或非笔：已有颜色与画笔颜色反相的组合）；12（复制笔或缺省值：画笔颜色）；13（或笔非：画笔颜色与已有颜色的反相的组合）；14（或笔：画笔颜色与已有颜色的组合）；15（白色：始终画出白色）。

例如：

```
画板_方式.画出方式 = 15
```

程序执行后将“画板\_方式”的画笔类型设为始终白色的画笔。

#### 9) “画笔粗细”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板中画笔的粗细。单位为“绘画单位”属性中的值，如果为0，表示画笔的粗细为一个像素点。当画笔类型为划线、点线、点划线、双点划线时，本属性无效，画笔宽度始终为一个像素点。

例如：

```
画板1.画笔粗细 = 10  
画板1.画直线 (0, 0, 100, 100)
```

程序执行后将“画板1”的画笔粗细设为10个单位，在画板1中从（0，0）到（100，100）画一条直线。

#### 10) “画笔颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板内画笔的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
画板_颜色.画笔颜色 = #蓝色
```

程序执行后将“画板\_颜色”的画笔颜色设为蓝色。

#### 11) “刷子类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于设置画板中刷子的类型。

本属性可以是以下值之一：0（空刷子）；1（颜色刷子）；2（右斜线）；3（直交叉线）；4（斜交叉线）；5（左斜线）；6（横线）；7（竖线）；8（5%色点）；9（10%色点）；10（20%色点）；11（25%色点）；12（30%色点）；13（40%色点）；14（50%色点）；15（60%色点）；16（70%色点）；17（75%色点）；18（80%色点）；19（90%色点）；20（浅色上对角线）；21（浅色下对角线）；22（深色上对角线）；23（深色下对角线）；24（宽上对角线）；25（宽下对角线）；26（浅色竖线）；27（浅色横线）；28（窄竖线）；29（窄横线）；30（深色竖线）；31（深色横线）；32（上对角虚线）；33（下对角虚线）；34（横虚线）；35（竖虚线）；36（小纸屑）；37（大纸屑）；38（之字形）；39（波浪线）；40（对角砖形）；41（横向砖形）；42（纺织物）；43（方格）；44（草皮）；45（虚格线）；46（点式菱形）；47（瓦形）；48（棚架）；49（球体）；50（小网格）；51（大网格）；52（小棋盘）；53（大棋盘）；54（空心菱形）；55（实心菱形）。

例如：

```
画板1.刷子类型 = 14
画板1.填充矩形 (0, 0, 100, 100)
```

程序执行后以半透明方式填充“画板1”的（0，0）～（100，100）区域。

### 12) “刷子颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
 本属性用于设置画板内刷子的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
画板_刷子.刷子颜色 = #绿色
```

程序执行后将“画板\_刷子”的刷子颜色设为绿色。

### 13) “文本颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
 本属性用于设置画板内写出文本的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

变量名	类 型	静态	数组	备 注
变量1	字体			

```
画板1.文本颜色 = #红色
画板1.文本背景颜色 = #黑色
变量1.加粗 = 真
变量1.倾斜 = 真
```





```

变量1.字体名称 = “隶书”
变量1.字体大小 = 36
画板1.字体 = 变量1
» 画板1.写文本行 (“易语言”)

```

程序执行后将“画板1”的文本颜色设为红色；将“画板1”的文本背景色设为黑色；改变“画板1”写出文字时的字体；在“画板1”中写出一行文本，内容为“易语言”。

#### 14) “文本背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板内写出文本的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```

画板1.文本颜色 = #红色
画板1.文本背景颜色 = #黑色
变量1.加粗 = 真
变量1.倾斜 = 真
变量1.字体名称 = “隶书”
变量1.字体大小 = 36
画板1.字体 = 变量1
» 画板1.写文本行 (“易语言”)

```

程序执行后将“画板1”的文本颜色设为红色；将“画板1”的文本背景色设为黑色；改变“画板1”写出文字时的字体；在“画板1”中写出一行文本，内容为“易语言”。

#### 15) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置画板内写出文本的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```

画板1.文本颜色 = #红色
画板1.文本背景颜色 = #黑色
变量1.加粗 = 真
变量1.倾斜 = 真
变量1.字体名称 = “隶书”
变量1.字体大小 = 36
画板1.字体 = 变量1
» 画板1.写文本行 (“易语言”)

```





程序执行后将“画板1”的文本颜色设为红色；将“画板1”的文本背景色设为黑色；改变“画板1”写出文字时的字体；在“画板1”中写出一行文本，内容为“易语言”。

#### 16) “画板宽度”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性保存了目前画板使用现行绘画单位下的宽度。属性值等于“画板”窗口组件的用户区域宽度换算为对应的单位值。

例如：

```
画板1.绘画单位 = 1
» 输出调试文本 (画板1.画板宽度)
» 输出调试文本 (画板1.画板高度)
```

程序运行后将“画板1”的绘画单位设为0.1毫米；在输出面板中显示“画板1”宽多少个单位；在输出面板中显示“画板1”高多少个单位。

#### 17) “画板高度”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性保存了目前画板使用现行绘画单位下的高度。属性值等于“画板”窗口组件的用户区域高度换算为对应的单位值。

例如：

```
画板1.绘画单位 = 1
» 输出调试文本 (画板1.画板宽度)
» 输出调试文本 (画板1.画板高度)
```

程序运行后将“画板1”的绘画单位设为0.1毫米；在输出面板中显示“画板1”宽多少个单位；在输出面板中显示“画板1”高多少个单位。

### 4.6.2.2 画板事件

“绘画”事件原型为：

返回值类型：无返回值

参数一：参数名：重画区左边 类型：整数型

参数二：参数名：重画区上边 类型：整数型

参数三：参数名：重画区右边 类型：整数型

参数四：参数名：重画区下边 类型：整数型

当画板“自动重画”属性为“假”，且画板中的全部或一部分区域需要被重新绘制时即触发本事件。用户程序在响应此事件期间在画板上所进行的任何绘制操作均被限制在“需重画区”参数所指定的区域内。

参数<1>指定了欲重画区域的起始横坐标(x)距离画板左边框的偏移值。

参数<2>指定了欲重画区域的起始纵坐标(y)距离画板顶边框的偏移值。





参数<3>指定了欲重画区域的结束横坐标 (x) 距离画板左边框的偏移值。

参数<4>指定了欲重画区域的结束纵坐标 (y) 距离画板顶边框的偏移值。

例如：

子程序名	返回值类型	公开	备 注		
_画板1_绘画					
参数名	类 型	参考	可空	数组	备 注
重画区左边	整数型				
重画区上边	整数型				
重画区右边	整数型				
重画区下边	整数型				

画板1. 填充矩形 (重画区左边, 重画区上边, 100, 100)

程序执行后当“画板1”接收到重画消息时触发“绘画”事件；在“画板1”需要重画的起始坐标处画一个100×100单位的实心矩形。

#### 4.6.2.3 画板命令

##### 1) “取设备句柄”命令

命令原型为：

<整数型> 对象. 取设备句柄 ()

如当前用户程序正在处理本画板所产生的“绘画”事件，本命令则用于获取画板所对应的设备句柄（即HDC），否则返回0。本命令一般配合Windows GDI类API使用。

例如：

子程序名	返回值类型	公开	备 注		
_画板1_绘画					
参数名	类 型	参考	可空	数组	备 注
重画区左边	整数型				
重画区上边	整数型				
重画区右边	整数型				
重画区下边	整数型				

变量名	类 型	静态	数组	备 注
rt	RECT			
hbr	整数型			

GetClientRect (画板1.取窗口句柄 (), rt)

hbr = CreateSolidBrush (#绿色)

FillRect (画板1.取设备句柄 (), rt, hbr)

DeleteObject (hbr)

程序执行后当触发“画板1”的“绘画”事件后将整个“画板1”填充为绿色。

本例程源码见随书光盘中“\图书例程\第四章\画板\取设备句柄.e”





## 2) “清除”命令

命令原型为:

<无返回值> 对象.清除 ([整数型 清除区左上角横坐标], [整数型 清除区左上角纵坐标], [整数型 清除区宽度], [整数型 清除区高度])

本命令用于清除画板上指定区域的内容并将当前文本写出位置移动到被清除区左上角。

参数<1>指定了欲清除区域左上角横坐标。使用当前绘画单位，相对于画板左上角。如果被省略，默认为0。

参数<2>指定了欲清除区域左上角纵坐标。使用当前绘画单位，相对于画板左上角。如果被省略，默认为0。

参数<3>指定了欲清除区域宽度。使用当前绘画单位。如果被省略，默认为清除区左边到画板右边之间的宽度。

参数<4>指定了欲清除区域高度。使用当前绘画单位。如果被省略，默认为清除区顶边到画板底边之间的高度。

例如:

画板1.填充矩形 (0, 0, 100, 100)

画板1.清除 (25, 25, 45, 45)

程序执行后从(0, 0)开始填充“画板1”100×100区域;清除“画板1”(25, 25)起45×45的区域。

## 3) “取点”命令

命令原型为:

<整数型> 对象.取点 (整数型 点横坐标, 整数型 点纵坐标)

本命令用于获取画板上指定点的颜色值。如果失败，将返回-1。

**注解:** 如果使用本命令时画板没有被显示，则本命令始终返回-1。

参数<1>指定了欲获取颜色点的横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了欲获取颜色点的纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如:

输出调试文本(画板\_取点.取点(10, 10))

程序执行后在输出面板中显示“画板\_取点”中(10, 10)点的颜色值。

## 4) “画点”命令

命令原型为:



<无返回值> 对象. 画点 (整数型 点横坐标, 整数型 点纵坐标, 整数型 欲画入点的颜色值)

本命令用于在画板指定位置使用指定颜色填充一个点。

参数<1>指定了欲填充点的横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了欲填充点的纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了用于填充点的颜色值。颜色值可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

画板\_画点.画点 (10, 10, #红色)

程序执行后在“画板\_画点”的 (10, 10) 坐标处使用红色填充一个点。

#### 5) “画直线”命令

命令原型为：

<无返回值> 对象. 画直线 (整数型 起始点横坐标, 整数型 起始点纵坐标, 整数型 结束点横坐标, 整数型 结束点纵坐标)

本命令用于使用当前画笔在画板上画出一条直线。

参数<1>指定了直线起始点横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了直线起始点纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了直线结束点横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了直线结束点纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_直线.画直线 (0, 0, 100, 100)

程序执行后在“画板\_直线”中从 (0, 0) 到 (100, 100) 画一条直线。

#### 6) “画椭圆”命令

命令原型为：

<无返回值> 对象. 画椭圆 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标)

本命令用于使用当前刷子在画板上填充一个椭圆。本命令的参数指定了外切于椭圆的矩形区域。





参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_椭圆.画椭圆 (0, 0, 100, 50)

程序执行后在“画板\_椭圆”中使用当前刷子填充内切于 (0, 0) , (100, 50) 矩形的椭圆。

#### 7) “画弧线”命令

命令原型为：

<无返回值> 对象.画弧线 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前画笔在画板上画出一条弧线。本命令所画图形相当于一个内切矩形的椭圆型边线的一部分。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于画板或打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于画板或打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于画板或打印纸左上角。





参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_弧线.画弧线 (0, 0, 100, 100, 0, 0, 50, 50)

执行效果如图4-30所示。

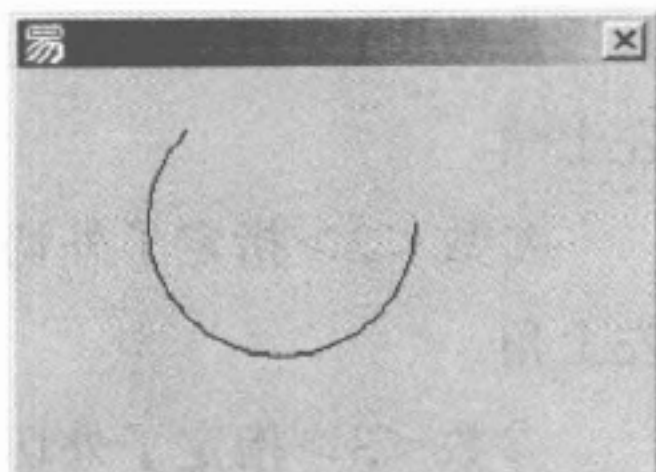


图4-30 画弧线效果

#### 8) “画弦” 命令

命令原型为：

<无返回值> 对象. 画弦 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前刷子在画板上填充一个椭圆的弦。椭圆的轮廓由外切矩形区域指定。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于画板或打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于画板或打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于画板或打印纸左上角。

参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_画弦.画弦 (0, 0, 100, 100, 0, 0, 50, 50)

执行效果如图4-31所示。

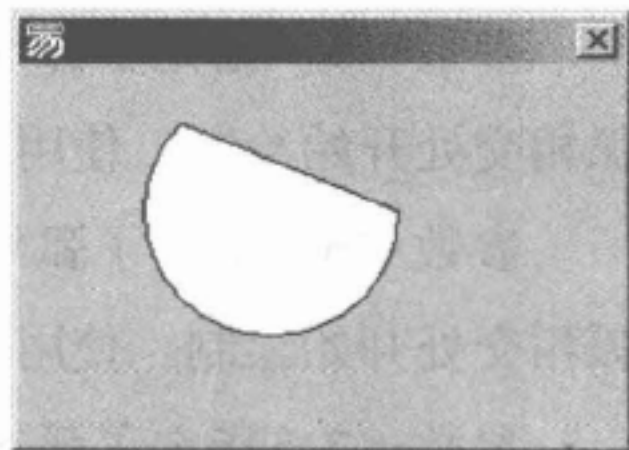


图4-31 画弦效果





### 9) “画饼”命令

命令原型为:

<无返回值> 对象. 画饼 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前刷子在画板上填充一个椭圆, 并切掉原切角区域(V型)。椭圆的轮廓由外切矩形区域指定。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位, 相对于画板或打印纸左上角。

参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位, 相对于画板或打印纸左上角。

例如:

画板\_画饼.画饼 (0, 0, 100, 100, 0, 0, 100, 50)

执行结果如图4-32所示。

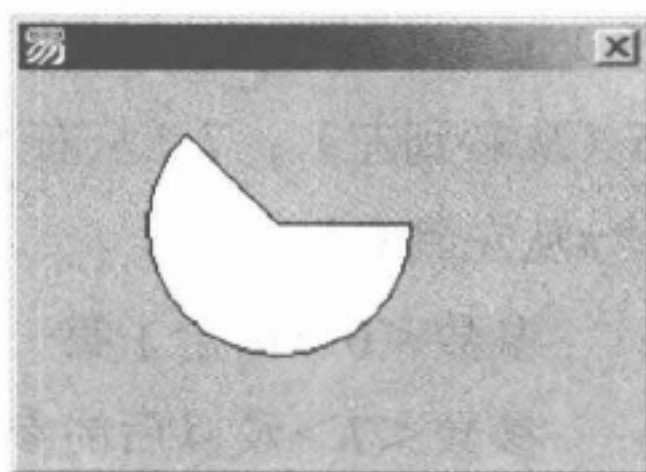


图4-32 画饼效果

### 10) “画矩形”命令

命令原型为:

<无返回值> 对象. 画矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标)

本命令用于使用当前画笔在画板上画出一个矩形, 矩形的内部使用当前刷子填充。

参数<1>指定了矩形左上角的横坐标。使用当前绘画单位, 相对于画板或打印纸左上角。





参数<2>指定了矩形左上角的纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了矩形右下角的横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了矩形右下角的纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_矩形.画矩形 (0, 0, 100, 100)

程序执行后在“画板\_矩形”中 (0, 0) 位置画一个100×100的矩形，并填充矩形区域。

#### 11) “画渐变矩形” 命令

命令原型为：

<无返回值> 对象. 画渐变矩形 (整数型 矩形区域左边, 整数型 矩形区域顶边, 整数型 矩形区域宽度, 整数型 矩形区域高度, [整数型 渐变方向], 整数型 首渐变颜色, 整数型 其他渐变颜色, ...)

本命令用于在画板上使用指定方式填充一个渐变矩形区域。本命令的最后一个参数可以重复扩展以指定更多渐变关键色。

参数<1>指定了欲填充矩形的起始横坐标。

参数<2>指定了欲填充矩形的起始纵坐标。

参数<3>指定了欲填充矩形的宽度。

参数<4>指定了欲填充矩形的高度。

参数<5>指定了形成渐变的样式。参数值可以为以下常量值之一：1（#从上到下）；2（#从左到右）；3（从左上到右下）；4（从右上到左下）；5（从下到上）；6（从右到左）；7（从右下到左上）；8（从左下到右上）。如果本参数被省略，默认为“#从左到右”。

参数<6>指定了第一个关键色的颜色值。

参数<7>及以后的参数指定了渐变时的关键颜色值。

例如：

画板\_矩形.画渐变矩形 (0, 0, 100, 100, #从左上到右下, #红色, #绿色, #蓝色)

执行效果如图4-33所示。

#### 12) “填充矩形” 命令

命令原型为：

<无返回值> 对象. 填充矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标)

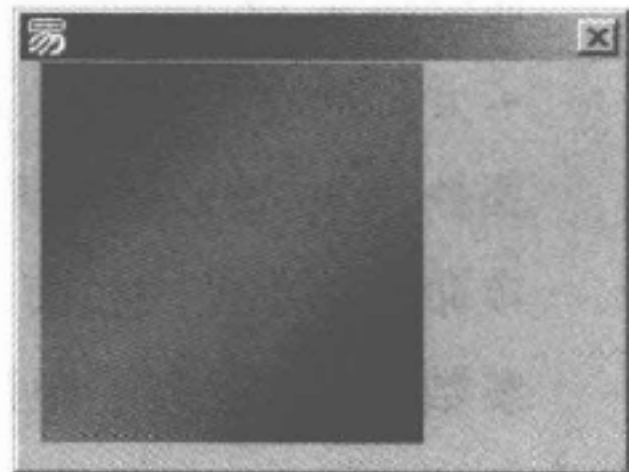


图4-33 画渐变矩形效果





本命令用于将画板上指定的矩形区域用当前刷子填充。本命令与“画矩形”命令区别在于有无边框。

参数<1>指定了欲填充矩形的左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了欲填充矩形的左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了欲填充矩形的右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了欲填充矩形的右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板\_填充.填充矩形 (0, 0, 100, 100)

程序执行后使用当前刷子填充“画板1”中的(0, 0)到(100, 100)的区域。

### 13) “画圆角矩形”命令

命令原型为：

<无返回值> 对象.画圆角矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标, 整数型 圆角宽度, [整数型 圆角高度])

本命令用于使用当前画笔在画板上画出一个圆角矩形，圆角矩形的内部使用当前刷子填充。

参数<1>指定了矩形的左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了矩形的左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了矩形的右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了矩形的右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<5>指定了圆角所在扇形的宽度。使用当前绘画单位。

参数<6>指定了圆角所在扇形的高度。使用当前绘画单位。如果省略本参数，默认与圆角宽度相等。



例如：

画板\_圆角.画圆角矩形 (0, 0, 100, 100, 20, 20)

执行效果如图4-34所示。

#### 14) “翻转矩形区” 命令

命令原型为：

<无返回值> 对象. 翻转矩形区 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标)

本命令用于将画板上指定矩形区域的颜色翻转过来。

参数<1>指定了矩形的左上角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了矩形的左上角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了矩形的右下角横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<4>指定了矩形的右下角纵坐标。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板1.画矩形 (0, 0, 100, 100)

画板1.翻转矩形区 (25, 25, 75, 75)

程序执行后，在“画板1”的 (0, 0) 至 (100, 100) 区域画一个矩形；翻转“画板1”中 (25, 25) 至 (75, 75) 的区域。

#### 15) “画多边形” 命令

命令原型为：

<无返回值> 对象. 画多边形 (整数型数组 多边形顶点, [整数型 顶点数目])

本命令用于在画板内使用当前画笔画一个多边形，并用当前刷子填充这个区域。如果所画的多边形没有闭合，将自动闭合。

参数<1>指定了多边形顶点坐标组。提供参数数据时只能提供数组数据，提供的数组在内部将自动转换为一维数组。数组结构为：顶点1横坐标，顶点1纵坐标，顶点2横坐标，顶点2纵坐标……以此类推。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了多边形顶点的总数目。如果被省略，默认为数组中所记录的顶点数目，既根据第二个参数成员数自动计算。

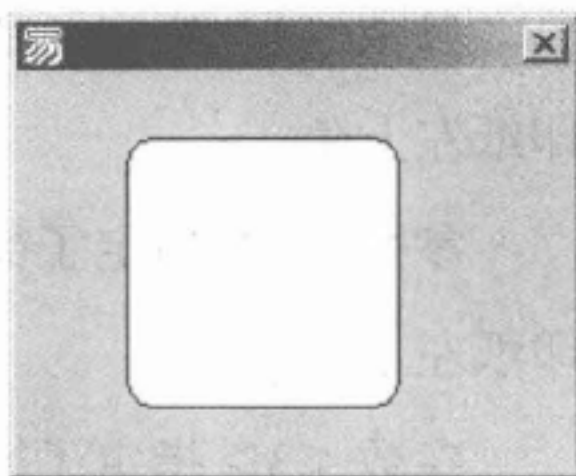


图4-34 画圆角矩形效果





例如：

变量名	类 型	静态	数组	备 注
变量1	整数型		3,2	

变量1 [1] [1] = 50

变量1 [1] [2] = 0

变量1 [2] [1] = 0

变量1 [2] [2] = 50

变量1 [3] [1] = 100

变量1 [3] [2] = 50

画板1.画多边形 (变量1, 3)

程序执行后，在“画板1”中画一个三角形。

### 16) “置写出位置” 命令

命令原型为：

<无返回值> 对象. 置写出位置 ( [整数型 横向写出位置] , [整数型 纵向写出位置] )

本命令用于设置下次使用“写文本行”或“写出”命令输出数据时的位置。

参数<1>指定了新写出位置的横坐标。如果本参数被省略，则使用现行横向写出位置。使用当前绘画单位，相对于画板或打印纸左上角

参数<2>指定了新写出位置的纵坐标。如果本参数被省略，则使用现行纵向写出位置。使用当前绘画单位，相对于画板或打印纸左上角。

例如：

画板1.置写出位置 (100, 100)

» 画板1.写文本行 (“易语言”)

程序执行后，设置新的文字写出位置为（100，100）；在“画板1”中使用当前字体及字体颜色写出“易语言”。

### 17) “写文本行” 命令

命令原型为：

<无返回值> 对象. 写文本行 ( [通用型 欲写出数据] , ... )

本命令用于在当前写出位置写出指定的文本、数值、逻辑值或日期时间，并将现行写出位置调整到下行行首。

参数<1>及后续参数指定了写出的内容。参数值只能为文本、数值、逻辑值或日期时间。如果本参数被省略，则写出一个空行。

例如：

画板1.置写出位置 (100, 100)

» 画板1.写文本行 (“易语言”)



程序执行后，设置新的文字写出位置为（100，100）；在“画板1”中使用当前字体及字体颜色写出“易语言”。

#### 18) “滚动写行”命令

命令原型为：

<无返回值> 对象. 滚动写行（[通用型 欲写出数据]，...）

本命令用于在当前写出位置写出指定的文本、数值、逻辑值或日期时间，并将现行写出位置调整到下行行首。如果现行画板高度无法容纳当前所要写出的行，则自动向上滚动画板内容。

参数<1>及后续参数指定了欲写出的内容。参数值只能为文本、数值、逻辑值或日期时间。如果本参数被省略，则写出一个空行。

例如：

画板\_写行.滚动写行（“易”，“语”，“言”）

程序执行后在“画板\_写行”中写出三行，依次为“易”、“语”、“言”。

#### 19) “写出”命令

命令原型为：

<无返回值> 对象. 写出（[通用型 欲写出数据]，...）

本命令用于在当前写出位置处写出指定的文本、数值、逻辑值或日期时间。自动调整现行写出位置到所写出数据的末位置。

参数<1>及后续参数指定了欲写出的内容。参数值只能为文本、数值、逻辑值或日期时间。如果本参数被省略，则不写出任何内容。

例如：

画板\_写出.写出（“易”，“语”，“言”）

程序执行后在“画板\_写出”当前写出位置分三次依次写出“易”、“语”、“言”。

#### 20) “定位写出”命令

命令原型为：

<无返回值> 对象. 定位写出（[整数型 横向写出位置]，[整数型 纵向写出位置]，通用型 欲写出数据，...）

本命令用于在指定写出位置处写出指定的文本、数值、逻辑值或日期时间，不改变现行写出位置。

参数<1>指定了写出文字的起始横坐标。如果本参数被省略，则使用现行横向写出位置。使用当前绘画单位，相对于画板或打印纸左上角。

参数<2>指定了写出文字的起始纵坐标。如果本参数被省略，则使用现行纵向写出





位置。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>及后续参数指定了欲写出的内容。参数值只能为文本、数值、逻辑值或日期时间。

例如：

画板\_定位.定位写出(50, 50, “易语言”)

程序执行后在“画板\_定位”的(50, 50)位置写出“易语言”。

## 21) “取宽度”命令

命令原型为：

<整数型> 对象.取宽度 (通用型 欲取其宽度或高度的数据)

本命令用于获取指定数据的在画板写出时的宽度，使用当前绘画单位。

参数<1>指定了欲测试的数据。参数值只能为文本、数值、逻辑值或日期时间。

例如：

输出调试文本(画板\_宽度.取宽度(“易语言”))

程序执行后在输出面板显示“易语言”在画板写出时所需要的宽度。

## 22) “取高度”命令

命令原型为：

<整数型> 对象.取高度 (通用型 欲取其宽度或高度的数据)

本命令用于获取指定数据在画板写出时的高度，使用当前绘画单位。

参数<1>指定了欲测试的数据。参数值只能为文本、数值、逻辑值或日期时间。

例如：

输出调试文本(画板\_高度.取高度(“易语言”))

程序执行后在输出面板显示“易语言”在画板写出时所需要的高度。

## 23) “画图片”命令

命令原型为：

<无返回值> 对象.画图片 (整数型 图片号, 整数型 图片左边画出位置, 整数型 图片顶边画出位置, [整数型 图片画出宽度], [整数型 图片画出高度], [整数型 图片画出方法])

本命令用于将使用“载入图片”命令所载入的资源以指定尺寸、方式画到画板指定位置。

参数<1>指定了使用“载入图片”命令所返回的图片号。

参数<2>指定了图片画出时位置的起始横坐标。使用当前绘画单位，相对于画板或打印纸左上角。

参数<3>指定了图片画出时位置的起始纵坐标。使用当前绘画单位，相对于画板或





打印纸左上角。

参数<4>指定了画出图片时将图片缩放到的目标宽度。如果本参数被省略，则使用图片本身的宽度。使用当前绘画单位

参数<5>指定了画出图片时将图片缩放到的目标高度。如果本参数被省略，则使用图片本身的高度。使用当前绘画单位

参数<6>定义在将图像绘制到对象（画板/打印机）上时对图像执行的位操作。对“图标”类型图片无效。可以为以下常量值之一或者其他自定义操作码：1（#拷贝）；2（#翻转拷贝）；3（#位异或）；4（#位或）；5（#位与）。

本参数也可以指定透明色，但必须是负颜色数值。如：

画板1.画图片 (图片号, 0, 0, , , -取颜色值 (255, 255, 255))

此语句的作用就是在画板上画出图片中除白色外的所有颜色。

如果省略本参数，默认为“#拷贝”。

例如：

变量名	类型	静态	数组	备注
变量1	整型			

变量1 = 载入图片 (“C:\WINDOWS\Prairie Wind.bmp”)

画板1.画图片 (变量1, 0, 0, , , #拷贝)

程序执行后将“C:\WINDOWS\Prairie Wind.bmp”画在画板（0，0）处。

#### 24) “取图片宽度”命令

命令原型为：

<整型> 对象. 取图片宽度 (整型 图片号)

本命令用于获取指定图片的宽度，使用当前绘画单位。

参数<1>指定了欲获取图片宽度对应的图片号。图片号由“载入图片”命令返回。

例如：

变量名	类型	静态	数组	备注
图片号	整型			

图片号 = 载入图片 (“C:\WINDOWS\Prairie Wind.bmp”)

→ 输出调试文本 (画板1.取图片宽度 (图片号))

程序执行后在输出面板中显示“C:\WINDOWS\Prairie Wind.bmp”转换为画板当前单位后的宽度。

#### 25) “取图片高度”命令

命令原型为：

<整型> 对象. 取图片高度 (整型 图片号)

本命令用于获取指定图片的高度，使用当前绘画单位。





参数<1>指定了欲获取图片宽度对应的图片号。图片号由“载入图片”命令返回。

例如：

变量名	类型	静态	数组	备注
图片号	整数型			

图片号 = 载入图片 (“C:\WINDOWS\Prairie Wind.bmp”)

→ 输出调试文本 (画板1.取图片高度 (图片号))

程序执行后在输出面板中显示“C:\WINDOWS\Prairie Wind.bmp”转换为画板当前单位后的高度。

## 26) “复制”命令

命令原型为：

<无返回值> 对象.复制 ([整数型 欲复制区域的左边], [整数型 欲复制区域的顶边], [整数型 欲复制区域的宽度], [整数型 欲复制区域的高度], [画板 复制到的目的画板], [整数型 欲复制到位置左边], [整数型 欲复制到位置顶边], [整数型 复制方法])

本命令用于将源画板（本命令的调用画板对象）中指定区域的内容快速复制到目的画板中的指定位置，使用源和目的画板各自的当前绘画单位。如果源画板当前不可视，其“自动重画”属性必须为真才有效。

参数<1>指定了欲复制区域起始横坐标。如果本参数被省略，默认值为0。使用源画板的当前绘画单位。

参数<2>指定了欲复制区域起始纵坐标。如果本参数被省略，默认值为0。使用源画板的当前绘画单位。

参数<3>指定了欲复制区域的宽度。如果本参数被省略，默认等同于源画板的宽度。使用源画板的当前绘画单位。

参数<4>指定了欲复制区域的高度。如果本参数被省略，默认等同于源画板的高度。使用源画板的当前绘画单位。

参数<5>指定了复制到的目的对象。如果本参数被省略，默认为复制到源画板对象。

参数<6>指定了复制到目的对象具体横坐标。如果本参数被省略，默认值为0。使用目的画板的当前绘画单位。

参数<7>指定了复制到目的对象具体纵坐标。如果本参数被省略，默认值为0。使用目的画板的当前绘画单位。

参数<8>定义了将在画板内容复制到目的画板上时对图像执行的位操作。

可以为以下常量值之一或者其他自定义操作码：1（#拷贝）；2（#翻转拷贝）；3（#位异或）；4（#位或）；5（#位与）。如果省略本参数，默认为“#拷贝”。



例如：

```
画板1.画矩形 (0, 0, 100, 100)
画板1.复制 (0, 0, 100, 100, 画板2, 0, 0, #拷贝)
```

程序执行后在“画板1”中(0, 0)位置画一个100×100的矩形；将“画板1”中(0, 0)～(100, 100)的矩形范围内的内容复制到“画板2”的(0, 0)处。

## 27) “取图片”命令

命令原型为：

<字节集> 对象. 取图片 ([整数型 输出宽度], [整数型 输出高度])

本命令用于获取画板上所有现有显示内容的图片数据。如果失败，返回空字节集。

参数<1>指定了图片的输出宽度。如果小于0，参数值指定的是最终图片输出宽度相对于所取得图片宽度的百分比（最小为10%）；如果等于0，则按图片原宽度输出；如果大于0，指定输出图片的绝对宽度。如果本参数被省略，默认值为0。

参数<2>指定了图片的输出高度。如果小于0，参数值指定的是最终图片输出高度相对于所取得图片高度的百分比（最小为10%）；如果等于0，则按图片原高度输出；如果大于0，指定输出图片的绝对高度。如果本参数被省略，默认值为0。

例如：

变量名	类型	静态	数组	备注
图片	字节集			

```
画板1.画矩形 (0, 0, 100, 100)
图片 = 画板1.取图片 (-50, -50)
图片框1.图片 = 图片
```

程序执行后将“画板1”中的内容缩小50%输出为图片数据；在“图片框1”中显示图片数据。

## 28) “单位转换”命令

命令原型为：

<整数型> 对象. 单位转换 (整数型 欲转换的坐标值, 整数型 欲转换坐标值的类型)

本命令用于将像素单位坐标值转换到当前绘画单位，或将当前绘画单位坐标值转换到像素单位。

参数<1>提供欲转换的横向或纵向坐标值。

参数<2>指定了上一个参数的类型。

参数可以为以下常量值之一： 1（#横向绘画单位）； 2（#纵向绘画单位）； 3（#横向像素单位）； 4（#纵向像素单位）。根据所提供的类型值，将进行相反的单位转换，即将绘画单位转换到像素单位，将像素单位转换到绘画单位。



例如：

```
画板1.绘画单位 = 1
➤ 输出调试文本 (画板1.单位转换 (50, #横向绘画单位))
```

程序执行后将“画板1”的绘画单位设为0.1毫米；将50个横向绘画单位转换为像素值显示在输出面板中。

### 4.6.3 颜色选择器

#### 4.6.3.1 颜色选择器属性

##### 1) “颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定颜色选择器所显示的现行颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
颜色选择器_颜色.颜色 = #绿色
程序执行后将“颜色选择器_颜色”的现行颜色设为绿色。
```

##### 2) “允许透明” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定运行后颜色选择器是否允许有透明色选项。“真”为有透明色选项，“假”为没有。

例如：

```
颜色选择器_透明.允许透明 = 真
程序执行后设置“颜色选择器_透明”拥有透明色选项。
```

#### 4.6.3.2 颜色选择器事件

“颜色被改变”事件原型为：

**返回值类型：**无返回值；**无参数**  
当颜色选择器的现行颜色被用户改变后将触发本事件。

例如：

子程序名	返回值类型	公开	备注
颜色选择器1_颜色被改变			

```
➤ 输出调试文本 (“颜色被改变”)
```

程序运行后，当“颜色选择器1”的现行颜色被选择后，在输出面板中显示“颜色被改变”。





## 4.7 分组类组件

### 4.7.1 分组框

#### 分组框属性

##### 1) “标题”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定分组框所显示的标题文字。如果希望某一字符成为本分组框中第一个可停留焦点组件的访问键，可以在该字符前面加上一个“&”字符。

例如：

```
分组框_标题.标题 = “易语言”
```

程序执行后会将“分组框\_标题”的标题改为“易语言”。

##### 2) “对齐方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定分组框标题的位置。本属性可以是以下值之一：0（左边）；1（居中）；2（右边）。

例如：

```
分组框1.标题 = “易语言”
```

```
分组框1.对齐方式 = 1
```

程序执行后会将“分组框1”的标题改为“易语言”；将“分组框1”的标题对齐方式设为居中显示。

##### 3) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定分组框标题的文本颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
分组框1.标题 = “易语言”
```

```
分组框1.文本颜色 = #黄色
```

程序执行后会将“分组框1”的标题改为“易语言”；将“分组框1”标题的文字颜色设为黄色。





4) “背景颜色” 属性

**属性类型：** 整数型； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于指定分组框的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

分组框\_背景.背景颜色 = #黑色  
程序执行后将“分组框\_背景”的背景颜色设为黑色。

5) “字体” 属性

**属性类型：** 字体； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于指定分组框标题所使用的字体属性。

例如：

变量名	类 型	静态	数组	备 注
变量1	字体			

分组框1.标题 = “易语言”  
变量1.加粗 = 真  
变量1.字体大小 = 36  
变量1.字体名称 = “黑体”  
分组框1.字体 = 变量1

程序执行后会将“分组框1”的标题改为“易语言”；改变标题所使用的字体。

4.7.2 外形框

外形框属性

1) “背景颜色” 属性

**属性类型：** 整数型； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于设置外形框内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

外形框\_背景.背景颜色 = #红色  
程序执行后会将“外形框\_背景”的背景颜色设为红色。

2) “外形” 属性

**属性类型：** 整数型； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于设置外型框的框形状。

本属性值可以是以下值之一：0（矩形）；1（正方形）；2（椭圆）；3（圆）；4（圆角矩形）；5（圆角正方形）；6（横向线）；7（纵向线）。





例如：

```
外形框_外形.外形 = 3
```

程序执行后会将“外形框\_外形”的框形改变为圆形。

### 3) “线条效果”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定外形框线条效果。

本属性可以是以下值之一：0（通常）；1（凹入）；2（凸出）。

例如：

```
外形框_效果.线条效果 = 2
```

程序执行后将“外形框\_效果”的线条效果设为凸出样式。

### 4) “线型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置外形框的线条类型。

本属性可以是以下值之一：0（无）；1（直线）；2（划线）；3（点线）；4（点划线）；5（双点划线）。

例如：

```
外形框_线型.线型 = 3
```

程序执行后会将“外形框\_线型”的边框线型设为点线样式。

### 5) “线宽”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置外形框线条的宽度，单位为像素。但仅在线条效果为“通常”且线型为“直线”时才有效。

例如：

```
外形框_线宽.线宽 = 5
```

程序执行后将“外形框\_线宽”的线条宽度设为5像素。

### 6) “线条颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置外形框线条的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。但仅在线条效果为“通常”时才有效。

例如：

```
外形框_线条.线条颜色 = #绿色
```

程序执行后将“外形框\_线条”的线条颜色设为绿色。





### 7) “填充颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置外形框中图形范围的填充颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。但仅在外形不为线条时才有效。

例如：

外形框\_填充色.填充颜色 = #蓝色

程序执行后将“外形框\_填充色”中图形填充为蓝色。

## 4.7.3 选择框

### 4.7.3.1 选择框属性

#### 1) “图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定显示在选择框上的图片。显示图片后选择框的标题将被隐藏。

例如：

选择框\_图片.图片 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“选择框\_图片”的图片。

#### 2) “按钮形式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框显示其选中状态的样式是否使用按钮样式。“真”为使用按钮样式，“假”为不使用。

例如：

选择框\_按钮.按钮形式 = 真

程序执行后开启“选择框\_按钮”的按钮样式效果。

#### 3) “平面”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框显示样式是否使用平面效果。“真”为使用，“假”为不使用。

例如：

选择框\_平面.平面 = 真

程序执行后开启“选择框\_平面”的平面效果。

#### 4) “标题”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定选择框所显示的标题文字。标题只有在图片为空时才有效，如果希望某一字符成为此选择框的访问键，可以在该字符前面加上一个 & 字符。

例如：

```
选择框_标题.标题 = “易语言”
```

程序执行后将“选择框\_标题”的标题设为“易语言”。

#### 5) “标题居左”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框的标题是否在选择项的左侧。“真”为在左侧，“假”为在右侧。

**注解：**当选择框“按钮形式”为“真”时，本属性不能体现实际效果。

例如：

```
选择框_标题.标题居左 = 真
```

程序执行后设置“选择框\_标题”标题在其选项的左侧。

#### 6) “横向对齐方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框内标题的横向对齐方式。

本属性可以是以下值之一：0（左边）；1（居中）；2（右边）。

例如：

```
选择框_对齐.横向对齐方式 = 0
```

程序执行后将“选择框\_对齐”的内容横向对齐方式设为左对齐。

#### 7) “纵向对齐方式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框内标题及选项的纵向对齐方式。

本属性可以是以下值之一：0（顶边）；1（居中）；2（底边）。

例如：

```
选择框_对齐.纵向对齐方式 = 0
```

程序执行后将“选择框\_对齐”的内容纵向对齐方式设为顶对齐。

#### 8) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择框标题的文本颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
选择框_文本.文本颜色 = #蓝色
```

程序执行后会将“选择框\_文本”的文本颜色设为蓝色。





## 9) “背景颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置选择框内的背景颜色（不包含选择项）。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
选择框_背景.背景颜色 = #红色
```

程序执行后会将“选择框\_背景”的背景颜色设为红色。

## 10) “字体” 属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置选择框内文字所使用的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

```
变量1.倾斜 = 真
```

```
变量1.字体名称 = “幼圆”
```

```
变量1.字体大小 = 30
```

```
选择框1.字体 = 变量1
```

程序执行后会为“选择框1”中的文字设置一个新的字体。

## 11) “数据源” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

```
选择框1.数据源 = “数据源1”
```

```
选择框1.数据列 = “gender”
```

程序执行后将“选择框1”的数据来源和“数据源1”绑定；将选择框显示内容与数据源当前记录中“gender”列绑定。

## 12) “数据列” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```
选择框1.数据源 = “数据源1”
```

```
选择框1.数据列 = “gender”
```



程序执行后将“选择框1”的数据来源和“数据源1”绑定；将选择框显示内容与数据源当前记录中“gender”列绑定。

13) “选中”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定选择框中项目是否选中。“真”为选中，“假”为不选中。

例如：

选择框\_选中.选中 = 真

程序执行后将“选择框\_选中”的选项选中。

4.7.3.2 选择框事件

“被单击”事件原型为：

无返回值；无参数

当鼠标单击选择框或焦点在该选择框并改变选中状态时触发本事件。

例如：

子程序名	返回值类型	公开	备注
_选择框1_被单击			

» 输出调试文本（“选择框被单击”）

程序运行后，当单击“选择框1”时在输出面板中显示“选择框被单击”。

4.7.4 单选框

4.7.4.1 单选框属性

1) “图片”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定显示在单选框上的图片。显示图片后单选框的标题将被隐藏。

例如：

单选框\_图片.图片 = 读入文件（“C:\WINDOWS\Prairie Wind.bmp”）

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“单选框\_图片”的图片。

2) “按钮形式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框显示其选中状态的样式是否使用按钮样式。“真”为使用按钮样式，“假”为不使用。

例如：

单选框\_按钮.按钮形式 = 真



程序执行后开启“单选框\_按钮”的按钮形式功能。

### 3) “平面”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框显示样式是否使用平面效果。“真”为使用，“假”为不使用。

例如：

```
单选框_平面.平面 = 真
```

程序执行后开启“单选框\_平面”的平面效果。

### 4) “标题”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定单选框所显示的标题文字。标题只有在图片为空时才有效，如果希望某一字符成为此单选框的访问键，可以在该字符前面加上一个“&”字符。

例如：

```
单选框_标题.标题 = “易语言”
```

程序执行后将“单选框\_标题”的标题设为“易语言”。

### 5) “标题居左”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框的标题是否在选择项的左侧。“真”为在左侧，“假”为在右侧。

**注：**当单选框“按钮形式”为“真”时，本属性不能体现实际效果。

例如：

```
单选框_标题.标题居左 = 真
```

程序执行后设置“单选框\_标题”标题在其选项的左侧。

### 6) “横向对齐方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框内标题的横向对齐方式。本属性可以是以下值之一：0（左边）；1（居中）；2（右边）。

例如：

```
单选框_对齐.横向对齐方式 = 0
```

程序执行后将“单选框\_对齐”的内容横向对齐方式设为左对齐。

### 7) “纵向对齐方式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框内标题及选项的纵向对齐方式。  
本属性可以是以下值之一：0（顶边）；1（居中）；2（底边）。



例如：

单选框\_对齐.纵向对齐方式 = 0

程序执行后将“单选框\_对齐”的内容纵向对齐方式设为顶对齐。

#### 8) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框标题的文本颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

单选框\_颜色.文本颜色 = #蓝色

程序执行后会将“单选框\_颜色”的文本颜色设为蓝色。

#### 9) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框内的背景颜色（不包含选择项）。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

单选框\_颜色.背景颜色 = #红色

程序执行后会将“单选框\_颜色”的背景颜色设为红色。

#### 10) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置单选框内文字所使用的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

变量1.删除线 = 真

变量1.角度 = 15

变量1.字体大小 = 10

单选框1.字体 = 变量1

程序执行后会为“单选框1”中的文字设置一个新的字体。

#### 11) “选中”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定单选框中项目是否选中。“真”为选中，“假”为不选中。

例如：

单选框\_选中.选中 = 真

程序执行后将“单选框\_选中”的选项选中。



#### 4.7.4.2 单选框事件

“被单击”事件原型为：

无返回值，无参数

当鼠标单击单选框或焦点在该单选框并改变选中状态时触发本事件。

例如：

子程序名	返回值类型	公开	备注
_单选框1_被单击			

→ 输出调试文本（“单选框被单击”）

程序运行后，当单击“单选框1”时在输出面板中显示“单选框被单击”。

#### 4.7.5 选择夹

##### 4.7.5.1 选择夹属性

1) “表头方向”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定子夹在选择夹中的位置。可供选择的属性值有：0（上）；1（下）；

2（左）；3（右）。

例如：

选择夹\_表头.表头方向 = 2

程序执行后设置“选择夹\_表头”中子夹在选择夹左侧。

2) “允许多行表头”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择夹是否允许以多行样式排列子夹。“真”为使用多行排列，

“假”为使用单行排列。

例如：

选择夹\_表头.允许多行表头 = 假

程序执行后将“选择夹\_表头”设置为单行显示所有子夹样式。

3) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置选择夹中子夹标题的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

变量1.字体名称 = “幼圆”

变量1.字体大小 = 30

选择夹1.字体 = 变量1



程序执行后设置“选择夹1”的子夹标题为新的字体。

#### 4) “子夹管理”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定各列表项目。

在设计时易语言提供了一个编辑器，方便我们修改本属性，点击“...”打开项目编辑器，如图4-35所示。

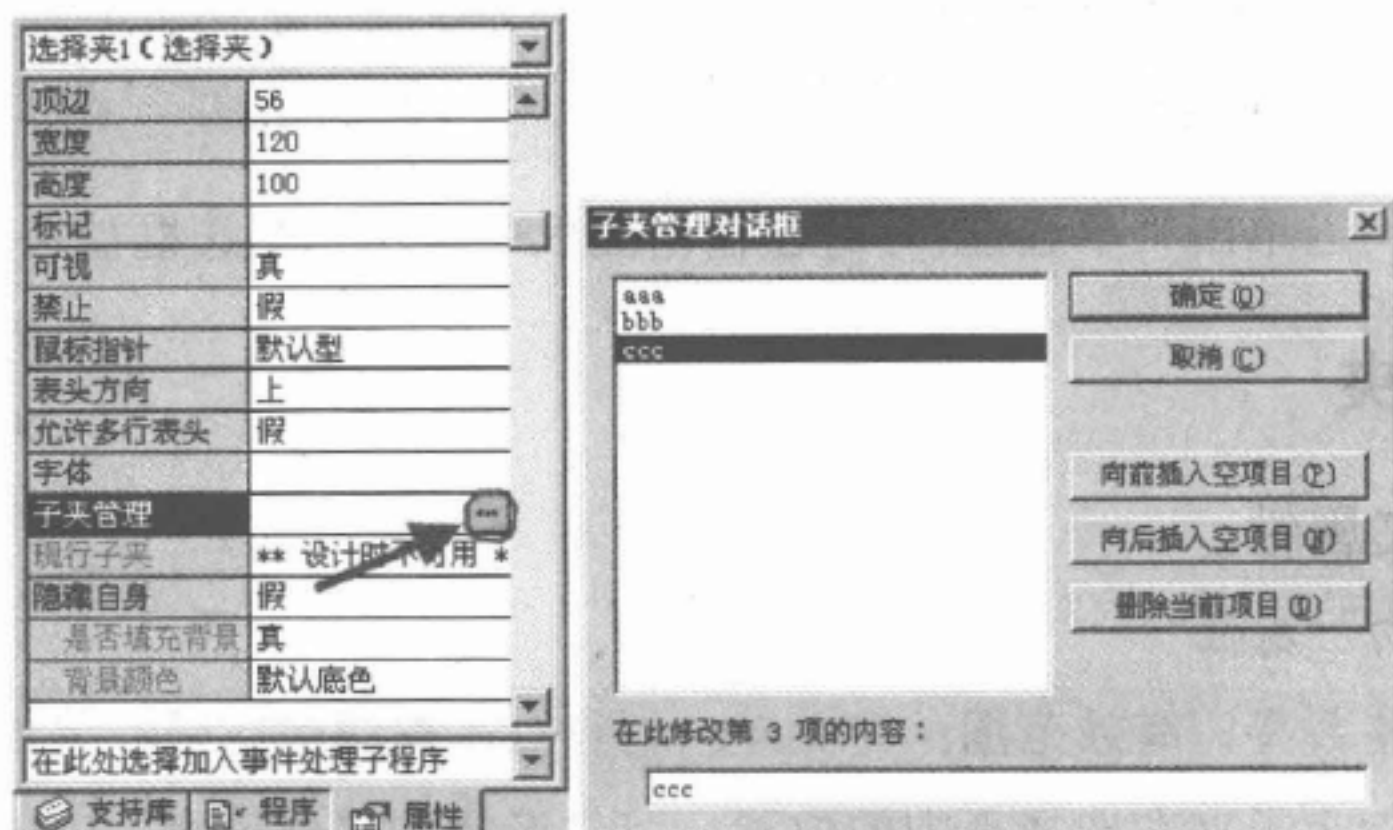


图4-35 打开子夹管理编辑器

本属性可以在编程时进行调用，使它在程序运行时可以动态的改变。  
本属性数据存储格式见表4-7。

表4-7 数据存储格式

数据类型 (长度)	说明	数据类型 (长度)	说明
短整数型	子夹数	整数型	第二个子夹文字长度 (字节数)
整数型	第一个子夹文字长度 (字节数)	字节型	第二个子夹文字的ASCII码数组
字节型	第一个子夹文字的ASCII码数组	.....	.....

例如：

```
选择夹1.子夹管理 = { 3, 0, 1, 0, 0, 0, 97, 2, 0, 0, 0, 98, 98,
3, 0, 0, 0, 99, 99, 99 }
选择夹1.现行子夹 = 2
```

程序执行后改变“选择夹1”的子夹为a, bb, ccc；选择第三个为现行子夹。  
本例程源码见随书光盘中“\图书例程\第四章\选择夹\子夹管理.e”

#### 5) “现行子夹”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取、更改  
本属性用于指定现行被选中子夹的索引，索引值从0开始。

例如：

输出调试文本 (选择夹1.现行子夹)



程序执行后，在输出面板中显示“选择夹1”当前子夹的索引号。

#### 6) “隐藏自身”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改（仅设计时）

本属性可以用来动态切换数个同一位置处子组件群的显示。此时选择夹本身不会被显示，但用户可以通过改变其“现行子夹”属性来显示指定子夹内的所有子组件。

**注解：**在设计时为方便用户定位子组件，表头方向始终为下。另外本属性不支持运行时修改。

例如：

```
--- 如果真 (选择夹1.隐藏自身 = 真)
    选择夹1.是否填充背景 = 真
    选择夹1.背景颜色 = #黄色
```

程序执行后判断“选择夹1”隐藏自身是否为“真”；若满足条件则允许填充背景并填充黄色作为背景色。

#### 7) “是否填充背景”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“隐藏自身”属性为“真”，本属性用于指定是否填充选择夹背景颜色。“隐藏自身”属性为“真”时如将本属性设置为假，可能会引起选择夹内一些子组件的刷新问题，此时将此属性设置为真并指定一个合适的背景颜色即可解决此问题。

例如：

```
--- 如果真 (选择夹1.隐藏自身 = 真)
    选择夹1.是否填充背景 = 真
    选择夹1.背景颜色 = #黄色
```

程序执行后判断“选择夹1”隐藏自身是否为“真”；若满足条件则允许填充背景并填充黄色作为背景色。

#### 8) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“隐藏自身”属性为“真”，本属性用于指定选择夹的背景填充颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
--- 如果真 (选择夹1.隐藏自身 = 真)
    选择夹1.是否填充背景 = 真
    选择夹1.背景颜色 = #黄色
```



程序执行后判断“选择夹1”隐藏自身是否为“真”；若满足条件则允许填充背景并填充黄色作为背景色。

#### 4.7.5.2 选择夹事件

##### 1) “被单击”事件

事件原型为：

无返回值，无参数

当选择夹中的子夹被单击时触发本事件。

例如：

子程序名	返回值类型	公开	备注
_选择夹1_被单击			

» 输出调试文本（“被单击”）

程序运行后，当单击“选择夹1”中的子夹时在输出面板中显示“被单击”。

##### 2) “将改变子夹”事件

事件原型为：

无返回值，无参数

当选择夹中的现行子夹将被改变前触发本事件。

例如：

子程序名	返回值类型	公开	备注
_选择夹1_将改变子夹	逻辑型		

» 输出调试文本（“将改变子夹”）

程序运行后，当“选择夹1”中的现行子夹被改变前在输出面板中显示“将改变子夹”。

##### 3) “子夹被改变”事件

事件原型为：

无返回值，无参数

当选择夹中的现行子夹被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备注
_选择夹1_子夹被改变			

» 输出调试文本（“子夹被改变”）

程序运行后，当“选择夹1”中的现行子夹被改变后在输出面板中显示“子夹被改变”。



### 4.7.5.3 选择夹命令

#### 1) “取子夹数目”命令

命令原型为:

<整数型> 对象. 取子夹数目 ( )

本命令用于获取选择夹中子夹的数量。

例如:

输出调试文本 (选择夹\_子夹数目.取子夹数目 ( ))

程序执行后在输出面板中显示“选择夹\_子夹数目”中的子夹数量。

#### 2) “取子夹名称”命令

命令原型为:

<文本型> 对象. 取子夹名称 (整数型 子夹索引)

本命令用于获取指定子夹的标题。

参数<1>指定了欲获取子夹的索引值。索引值从1开始，1 代表子夹一，2 代表子夹二，以此类推。

例如:

选择夹1. 置子夹名称 (1, “易语言”)

» 输出调试文本 (选择夹1. 取子夹名称 (1))

程序执行后设置“选择夹1”中的第一个子夹标题为“易语言”；在输出面板中显示“选择夹1”中第一个子夹的标题。

#### 3) “置子夹名称”命令

命令原型为:

<逻辑型> 对象. 置子夹名称 (整数型 子夹索引, 文本型 欲置入的子夹名称)

本命令用于设置指定子夹的标题。

参数<1>指定了欲设置标题的子夹的子夹索引。索引值从1开始，1 代表子夹一，2 代表子夹二，以此类推。

参数<2>指定了新标题文本。

例如:

选择夹1. 置子夹名称 (1, “易语言”)

» 输出调试文本 (选择夹1. 取子夹名称 (1))

程序执行后设置“选择夹1”中的第一个子夹标题为“易语言”；在输出面板中显示“选择夹1”中第一个子夹的标题。





## 4.8 位置控制类组件

### 4.8.1 进度条

#### 进度条属性

##### 1) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定进度条边框样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

进度条\_边框.边框 = 5

程序执行后将“进度条\_边框”的边框样式设为单线边框式。

##### 2) “方向”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定进度条的样式。可供选择的属性值有：0（横向）；1（纵向）。

例如：

进度条\_方向.方向 = 1

程序执行后将进度条显示方式设为纵向进度条。

##### 3) “显示方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定进度条显示进度的样式。可供选择的属性值有：0（分块）；1（连续）。

例如：

进度条\_显示.显示方式 = 1

程序执行后将“进度条\_显示”的进度显示样式设为连续条状。

##### 4) “最小位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定进度条“位置”属性下限值。

例如：

进度条\_位置.最小位置 = 10

程序执行后将“进度条\_位置”的位置下限值设为10。



## 5) “最大位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定进度条“位置”属性上限值。

例如：

进度条\_位置.最大位置 = 20

程序执行后将“进度条\_位置”的位置上限值设为20。

## 6) “位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设定进度条的现行位置。

例如：

进度条\_位置.位置 = 15

程序执行后将“进度条\_位置”的现行位置设为15。

## 4.8.2 滑块条

### 4.8.2.1 滑块条属性

## 1) “边框” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滑块条边框样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

滑块条\_边框.边框 = 5

程序执行后将“滑块条\_边框”的边框设为单线边框样式。

## 2) “方向” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滑块条滑动的方向。可供选择的属性值有：0（横向）；1（纵向）。

例如：

滑块条\_方向.方向 = 1

程序执行后将“滑块条\_方向”的滑动方向设为纵向滑动。

## 3) “刻度类型” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滑块条刻度的样式。可供选择的属性值有：0（无）；1（上/左）；2（下/右）；3（双向）。





例如：

```
滑块条_刻度.刻度类型 = 3
```

程序执行后将“滑块条\_刻度”的刻度样式设为双向刻度。

#### 4) “单位刻度值”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“刻度类型”属性不为0，本属性用于设置每个刻度的单位值。

例如：

```
滑块条_刻度.单位刻度值 = 2
```

程序执行后将“滑块条\_刻度”的每刻度单位值设为2。

#### 5) “允许选择”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性设置是否开启滑块条选中样式。“真”为开启，“假”为关闭。

例如：

```
滑块条1.允许选择 = 真
```

```
滑块条1.首选择位置 = 3
```

```
滑块条1.选择长度 = 5
```

程序运行后开启“滑块条1”的允许选择功能；选中从第3个位置起5个单位。

#### 6) “首选择位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许选择”属性为“真”，本属性用于设置滑块条起始选中位置。

例如：

```
滑块条1.允许选择 = 真
```

```
滑块条1.首选择位置 = 3
```

```
滑块条1.选择长度 = 5
```

程序运行后开启“滑块条1”的允许选择功能；选中从第3个位置起5个单位。

#### 7) “选择长度”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许选择”属性为“真”，本属性用于设置选中的长度。

例如：

```
滑块条1.允许选择 = 真
```

```
滑块条1.首选择位置 = 3
```

```
滑块条1.选择长度 = 5
```

程序运行后开启“滑块条1”的允许选择功能；选中从第3个位置起5个单位。





## 8) “页改变值” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当用户在空隙处单击后滑块条位置的增减数值。

例如：

滑块条\_变值.页改变值 = 3

程序执行后将“滑块条\_变值”的页改变值设为3个单位。

## 9) “行改变值” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当用户拖动滑块时滑块条位置的每次增减数值。

例如：

滑块条\_变值.行改变值 = 2

程序执行后将“滑块条\_变值”的行改变值设为2个单位。

## 10) “最小位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置“位置”属性的下限值。

例如：

滑块条\_位置.最小位置 = -5

程序执行后将“滑块条\_位置”位置属性下限值设为“-5”。

## 11) “最大位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置“位置”属性的上限值。

例如：

滑块条\_位置.最大位置 = 5

程序执行后将“滑块条\_位置”位置属性上限值设为“5”。

## 12) “位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设定滑块条的现行位置。

例如：

滑块条\_位置.位置 = 0

程序执行后将“滑块条\_位置”的现行滑块位置设为“0”。

## 4.8.2.2 滑块条事件

“位置被改变”事件的原型为：

无返回值，无参数

当滑动条的滑块被拖动则触发本事件。





例如：

子程序名	返回值类型	公开	备注
_滑块条1_位置被改变			

» 输出调试文本（“位置被改变”）

程序执行后，当拖动“滑块条1”中的滑块时在输出面板中显示“位置被改变”。

### 4.8.3 横向滚动条

#### 4.8.3.1 横向滚动条属性

##### 1) “最小位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滚动条“位置”属性下限值。

例如：

横向滚动条\_位置.最小位置 = 10

程序执行后将“横向滚动条\_位置”的位置下限值设为10。

##### 2) “最大位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滚动条“位置”属性上限值。

例如：

横向滚动条\_位置.最大位置 = 20

程序执行后将“横向滚动条\_位置”的位置上限值设为20。

##### 3) “页改变值”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当用户在滚动条的空隙处单击后滚动条位置的增减数值。

例如：

横向滚动条\_变值.页改变值 = 5

程序执行后将“横向滚动条\_变值”的“页改变值”设为5。

##### 4) “行改变值”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当用户按下滚动条的左右箭头按钮后滚动条位置的增减数值。

例如：

横向滚动条\_变值.行改变值 = 2

程序执行后将“横向滚动条\_变值”的“行改变值”设为2。





### 5) “位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设定滚动条的现行位置。

例如：

横向滚动条\_位置.位置 = 15

程序执行后将“横向滚动条\_位置”的现行位置设为15。

### 6) “允许拖动跟踪” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设定当用户拖动滚动条的位置盒的过程中是否产生位置改变事件。“真”为产生事件，“假”为不产生事件。

例如：

横向滚动条\_跟踪.允许拖动跟踪 = 假

程序执行后设置拖动“横向滚动条\_跟踪”的位置盒过程中不产生位置被改变事件。

## 4.8.3.2 横向滚动条事件

“位置被改变”事件原型为：

无返回值，无参数

当横向滚动条的位置盒被拖动则触发本事件。

例如：

子程序名	返回值类型	公开	备注
_横向滚动条1_位置被改变			

» 输出调试文本（“位置被改变”）

程序执行后，当拖动“横向滚动条1”中的位置盒时在输出面板中显示“位置被改变”。

## 4.8.4 纵向滚动条

### 4.8.4.1 纵向滚动条属性

#### 1) “最小位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定滚动条“位置”属性下限值。

例如：

纵向滚动条\_位置.最小位置 = 10

程序执行后将“纵向滚动条\_位置”的位置下限值设为10。

#### 2) “最大位置” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定滚动条“位置”属性上限值。

例如：

纵向滚动条\_位置.最大位置 = 20

程序执行后将“纵向滚动条\_位置”的位置上限值设为20。

### 3) “页改变值”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定当用户在滚动条的空隙处单击后滚动条位置的增减数值。

例如：

纵向滚动条\_变值.页改变值 = 5

程序执行后将“纵向滚动条\_变值”的“页改变值”设为5。

### 4) “行改变值”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定当用户按下滚动条的左右箭头按钮后滚动条位置的增减数值。

例如：

纵向滚动条\_变值.行改变值 = 2

程序执行后将“纵向滚动条\_变值”的“行改变值”设为2。

### 5) “位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设定滚动条的现行位置。

例如：

纵向滚动条\_位置.位置 = 15

程序执行后将“纵向滚动条\_位置”的现行位置设为15。

### 6) “允许拖动跟踪”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设定当用户拖动滚动条的位置盒的过程中是否产生位置改变事件。“真”为产生事件，“假”为不产生事件。

例如：

纵向滚动条\_跟踪.允许拖动跟踪 = 假

程序执行后设置拖动“纵向滚动条\_跟踪”的位置盒过程中不产生位置被改变事件。

## 4.8.4.2 纵向滚动条事件

“位置被改变”事件原型为：

无返回值；无参数

当纵向滚动条的位置盒被拖动则触发本事件。





例如：

子程序名	返回值类型	公开	备注
_纵向滚动条1_位置被改变			

» 输出调试文本（“位置被改变”）

程序执行后，当拖动“纵向滚动条1”中的位置盒时在输出面板中显示“位置被改变”。

## 4.8.5 调节器

### 4.8.5.1 调节器属性

#### 1) “方向”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定调节器按钮指示方向。可供选择的属性值有：0（横向）；1（纵向）。

例如：

调节器\_方向.方向 = 0

程序执行后设置“调节器\_方向”的指示方向为横向调节样式。

#### 2) “热点跟踪”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定当鼠标移动到组件上时是否显示热点状态。“真”为开启，“假”为不开启。

例如：

调节器\_跟踪.热点跟踪 = 真

程序执行后开启“调节器\_跟踪”的热点跟踪功能。

### 4.8.5.2 调节器事件

“调节钮被按下”事件原型为：

**返回值类型：**无返回值

**参数一：**参数名：按钮值 类型：整数型

当程序执行后按下调节器上的按钮时触发本事件。

参数<1>保存了被按下的按钮值。如果按下的是调节器的向上箭头或向左箭头按钮，参数值为1，按下的是向下箭头或向右箭头为-1。

例如：

子程序名	返回值类型	公开	备 注		
_调节器1_调节钮被按下					
参数名	类 型	参考	可空	数组	备 注
按钮值	整数型				

变量名	类型	静态	数组	备注
变量	整数型	✓		





```
-- 如果 (按钮值 = 1)
-- 变量 = 变量 - 1
-- 变量 = 变量 + 1
-- 输出调试文本 (变量)
```

程序执行后，当按下“调节器1”的调节按钮时进行“变量”的增减操作，并在输出面板中显示当前“变量”的值。

## 4.9 网络与通信组件

### 4.9.1 客户

“客户”是面向连接（连接导向）的网络数据交换组件。“客户”组件需要和“服务器”组件建立连接后才能传输数据。基于连接的数据交换是一种可靠的、允许大数据量的网络数据交互方式。

#### 4.9.1.1 客户事件

##### 1) “数据到达”事件

事件原型为：

返回值类型：无返回值；无参数

当服务器端将数据发送过来后，会产生本事件。在本事件的处理子程序中调用“取回数据”命令即可取回本次所收到的数据。

例如：

子程序名	返回值类型	公开	备注
_客户1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			

收到的数据 = 客户1.取回数据 ()

→ 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))

当“客户1”与服务器成功连接后，接收到数据时在输出面板中显示收到的数据内容。

##### 2) “连接断开”事件

事件原型为：

返回值类型：无返回值；无参数

当连接被服务器端断开后，会产生本事件。





例如：

子程序名	返回值类型	公开	备注
_客户1_连接断开			

» 输出调试文本（“与服务器间的连接已断开”）

当“客户1”与已连接的服务器端断开时，在输出面板中显示“与服务器间的连接已断开”。

#### 4.9.1.2 客户命令

##### 1) “连接”命令

命令原型为：

<逻辑型> 对象. 连接（文本型 服务器地址，整数型 服务器端口号）

本命令用于连接到指定主机上的指定端口，该主机上的该端口必须已经被某一服务器组件监听。成功返回“真”，失败返回“假”。

参数<1>指定了欲连接的主机名或IP地址。

参数<2>指定了欲连接目标的端口号。

例如：

客户\_连接.连接（“127.0.0.1”，2010）

程序执行后“客户\_连接”会尝试连接本机的2010端口。

##### 2) “断开连接”命令

命令原型为：

<无返回值> 对象. 断开连接（）

本命令用于断开与服务器的已有连接。

例如：

客户\_连接.断开连接（）

程序执行后断开“客户\_连接”的已有连接。

##### 3) “发送数据”命令

命令原型为：

<逻辑型> 对象. 发送数据（通用型 欲发送数据）

本命令用于在成功建立与服务器的连接后，发送数据到服务器端组件。成功返回“真”，失败返回“假”。

参数<1>指定了欲发送的数据。

例如：

客户\_数据.发送数据（“易语言”）

程序执行后“客户\_数据”向已连接的服务器端发送内容为“易语言”的数据包。





## 4) “取回数据”命令

命令原型为：

<字节集> 对象. 取回数据 ()

本命令用于取回所接收到的数据。本命令必须在“数据到达”事件的处理子程序中使用。

例如：

子程序名	返回值类型	公开	备注
_客户1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			

收到的数据 = 客户1.取回数据 ()

→ 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))

当“客户1”收到服务器端发送来的数据时，在输出面板中显示所接收到的数据内容。

## 4.9.2 服务器

服务器是面向连接（连接导向）的网络数据交换的服务方组件。服务器需要在指定端口等待客户的连接，连接后才能与客户进行通信。

### 4.9.2.1 服务器属性

“端口”属性

**属性类型：**整数型，**有效范围：**设计时、编程时，**编程时权限：**使用、读取、更改  
本属性用于指定监听数据到达的端口号，可以是大于 0 小于 32767 的任何自定数值。

例如：

服务器\_端口.端口 = 2010

程序执行后设置“服务器\_端口”监听2010端口，等待客户端的连接。

### 4.9.2.2 服务器事件

#### 1) “数据到达”事件

事件原型为：

返回值类型：无返回值；无参数

当有数据到达后，会产生本事件。在本事件的处理子程序中调用“取回数据”命令即可取回本次所收到的数据。

例如：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
客户端信息	文本型		

涉及的程序集变量定义如上。





子程序名	返回值类型	公开	备注
_服务器1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			

收到的数据 = 服务器1.取回数据 ()

» 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))  
 服务器1.发送数据 (客户端信息, “已收到发送的数据”, )  
 服务器1.断开客户 (客户端信息)

当“服务器1”接收到从已连接的客户端发来的数据时，在输出面板中显示接收到的数据内容；向已连接的客户发送一段文本数据；断开与刚才发送数据的客户端的连接。

## 2) “客户进入”事件

事件原型为：

返回值类型：无返回值；无参数

当有新客户连入本服务器组件后，会产生本事件。在本事件的处理子程序中调用“取回客户”命令即可取回此新客户的地址（IP地址 + 端口）。

例如：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
客户端信息	文本型		

涉及的程序集变量定义如上。

子程序名	返回值类型	公开	备注
_服务器1_客户进入			

客户端信息 = 服务器1.取回客户 ()

程序执行后，当有客户端连接到“服务器1”时，将客户的信息取出并保存到“客户端信息”变量中。

## 3) “客户离开”事件

事件原型为：

返回值类型：无返回值；无参数

当有已连接客户断开与本服务器组件的连接后，会产生本事件。在本事件的处理子程序中调用“取回客户”命令即可取回此客户的地址（IP地址 + 端口）。

例如：

子程序名	返回值类型	公开	备注
_服务器1_客户离开			

» 输出调试文本 (“已断开与 ” + 服务器1.取回客户 () + “ 的连接”)





程序执行后，当与“服务器1”已建立连接的客户端主动断开连接时，在输出面板中显示断开的客户端信息。

#### 4.9.2.3 服务器命令

##### 1) “取回数据命令

命令原型为：

<字节集> 对象. 取回数据 ()

本命令用于取回所接收到的数据，本命令必须在“数据到达”事件的处理子程序中使用。

例如：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
客户端信息	文本型		

涉及的程序集变量定义如上。

子程序名	返回值类型	公开	备 注
_服务器1_数据到达			

变量名	类 型	静态	数组	备 注
收到的数据	字节集			

收到的数据 = 服务器1. 取回数据 ()

» 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))  
 服务器1. 发送数据 (客户端信息, “已收到发送的数据”, )  
 服务器1. 断开客户 (客户端信息)

当“服务器1”接收到从已连接的客户端发来的数据时，在输出面板中显示接收到的数据内容；向已连接的客户发送一段文本数据；断开与刚才发送数据的客户端的连接。

##### 2) “取回客户”命令

命令原型为：

<文本型> 对象. 取回客户 ()

本命令用于当接收到“客户进入”、“客户离开”或“数据到达”事件时，在该事件的处理子程序中可调用本命令取回对应的客户地址（IP地址+端口）。

例如：

子程序名	返回值类型	公开	备注
_服务器1_客户离开			

» 输出调试文本 (“已断开与 ” + 服务器1. 取回客户 () + “ 的连接”)

程序执行后，当与“服务器1”已建立连接的客户端主动断开连接时，在输出面板中显示断开的客户端信息。





3) “发送数据” 命令

命令原型为:

<逻辑型> 对象. 发送数据 (文本型 接收客户, 通用型 欲发送数据, [整数型 最长等待时间] )

本命令用于向指定已经连接进来的客户发送数据。成功返回真，失败返回假。

参数<1>指定了欲发送数据的目的。本参数必需为“取回客户”命令所返回的客户地址文本。

参数<2>指定欲发送到客户端的数据。

参数<3>指定等待发送成功的最长时间，单位为秒。如果省略本参数，默认为无限等待。

例如:

窗口程序集名	保 留	保 留	备 注
窗口程序集1			
变量名	类 型	数 组	备 注
客户端信息	文本型		

涉及的程序集变量定义如上。

子程序名	返回值类型	公开	备 注
_服务器1_数据到达			

变量名	类 型	静态	数 组	备 注
收到的数据	字节集			

收到的数据 = 服务器1.取回数据 ()

» 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))  
服务器1.发送数据 (客户端信息, “已收到发送的数据”, )  
服务器1.断开客户 (客户端信息)

当“服务器1”接收到从已连接的客户端发来的数据时，在输出面板中显示接收到的数据内容；向已连接的客户发送一段文本数据；断开与刚才发送数据的客户端的连接。

4) “断开客户” 命令

命令原型为:

<无返回值> 对象. 断开客户 (文本型 欲断开客户)

本命令用于断开与指定客户之间的连接。

例如:

窗口程序集名	保 留	保 留	备 注
窗口程序集1			
变量名	类 型	数 组	备 注
客户端信息	文本型		

涉及的程序集变量定义如上。



子程序名	返回值类型	公开	备注
_服务器1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			

收到的数据 = 服务器1.取回数据 ()

» 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))

服务器1.发送数据 (客户端信息, “已收到发送的数据”, )

服务器1.断开客户 (客户端信息)

当“服务器1”接收到从已连接的客户端发来的数据时，在输出面板中显示接收到的数据内容；向已连接的客户发送一段文本数据；断开与刚才发送数据的客户端的连接。

### 4.9.3 数据报

数据报是一种不可靠、小数据量的网络数据交互方式。如果传递的数据量过大，有可能会失败，最好不要超过 127 bit。如需要大数据量或者可靠数据传送方式，请使用面向连接的其他网络组件。数据报的优势在于无需建立连接，可批量群发，速度较快。劣势主要为不能保证发送的数据确实的到达目标。

#### 4.9.3.1 数据报属性

“端口”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定监听数据到达的端口号，可以是大于 0 小于 32767 的任何自定数值。  
例如：

数据报1.端口 = 2010

数据报1.发送数据 (“127.0.0.1”, 2010, “易语言”)

程序执行后将“数据报1”所监听的端口设为“2010”；向本机的2010端口发送数据报，内容为“易语言”。

#### 4.9.3.2 数据报事件

“数据到达”事件原型为：

**返回值类型：**无返回值；**无参数**

当有数据到达后，会产生本事件。在本事件的处理子程序中调用“取回数据”命令即可取回本次所收到的数据。

例如：

子程序名	返回值类型	公开	备注
_数据报1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			





```
收到的数据 = 数据报1.取回数据 ()
⇨ 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))
```

当“数据报1”收到数据时，在输出面板中显示收到的数据。

#### 4.9.3.3 数据报命令

##### 1) “发送数据”命令

命令原型为：

```
<逻辑型> 对象.发送数据 ([文本型 接收主机地址], 整数型 接收主机端口号,
通用型 欲发送数据)
```

本命令用于发送数据到指定主机上的指定端口。成功返回“真”，失败返回“假”。

参数<1>指定了接收数据的主机地址。该参数可以为主机名、IP地址等。如果省略本参数或者提供空文本，则在指定端口广播欲发送数据。

参数<2>指定了接收数据主机监听的端口号。

参数<3>指定了欲发送的数据。欲发送数据必须是系统基本数据类型。

例如：

```
数据报1.端口 = 2010
数据报1.发送数据 ("127.0.0.1", 2010, "易语言")
```

程序执行后将“数据报1”所监听的端口设为“2010”；向本机的2010端口发送数据报，内容为“易语言”。

##### 2) “取回数据”命令

命令原型为：

```
<字节集> 对象.取回数据 ()
```

本命令用于取回所接收到的数据。本命令必须在“数据到达”事件的处理子程序中使用。

例如：

子程序名	返回值类型	公开	备注
_数据报1_数据到达			

变量名	类型	静态	数组	备注
收到的数据	字节集			

```
收到的数据 = 数据报1.取回数据 ()
⇨ 输出调试文本 (取字节集数据 (收到的数据, #文本型, ))
```

当“数据报1”收到数据时，在输出面板中显示收到的数据。





## 4.9.4 超级链接框

### 4.9.4.1 超级链接框属性

#### 1) “标题”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置超级链接框中所显示的文字。

超级链接框\_标题.标题 = “易语言”

程序执行后将“超级链接框\_标题”中的文字设为“易语言”。

#### 2) “类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定超级链接框的链接类型。可供选择的属性值有：0（电子信箱地址）；  
1（Internet地址）。

例如：

超级链接框\_类型.类型 = 1

程序执行后设置“超级链接框\_类型”的链接类型为Internet地址。

#### 3) “电子信箱地址”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
如果“类型”属性值为“0”时。本属性用于指定链接到的目标邮箱地址。程序运行后，点击标题后会自动起用电子邮箱软件，并填写发信目标为本属性指定的地址。

例如：

超级链接框\_地址.电子信箱地址 = “sale@eflysky.com”

程序执行后设置“超级链接框\_地址”的电子信箱目标地址为“sale@eflysky.com”。

#### 4) “Internet地址”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
如果“类型”属性值为“1”时。本属性用于指定链接到的目标Internet地址。程序运行后，点击标题后会自动跳转到本属性所指定的地址。

例如：

超级链接框\_地址.Internet地址 = “http://bbs.eyuyan.com”

程序执行后将“超级链接框\_地址”的Internet链接目标地址为“http://bbs.eyuyan.com”。

#### 5) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定正常情况下（未点击）时的标题文本显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。





例如：

```
超级链接框1.标题 = “易语言”  
超级链接框1.类型 = 1  
超级链接框1.Internet地址 = “http://www.eyuyan.com”  
超级链接框1.文本颜色 = #绿色  
超级链接框1.访问后的颜色 = #红色  
超级链接框1.热点颜色 = #黄色  
超级链接框1.背景颜色 = #黑色
```

程序执行后设置“超级链接框1”的标题为“易语言”；链接类型为Internet地址；链接目标为“http://www.eyuyan.com”；未访问时的文本颜色设为绿色；访问后的文本颜色为“红色”；鼠标移到标题上时标题的热点颜色为黄色；整个超级链接框的背景色设为黑色。

#### 6) “访问后的颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定被访问过后的标题文本显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
超级链接框1.标题 = “易语言”  
超级链接框1.类型 = 1  
超级链接框1.Internet地址 = “http://www.eyuyan.com”  
超级链接框1.文本颜色 = #绿色  
超级链接框1.访问后的颜色 = #红色  
超级链接框1.热点颜色 = #黄色  
超级链接框1.背景颜色 = #黑色
```

程序执行后设置“超级链接框1”的标题为“易语言”；链接类型为Internet地址；链接目标为“http://www.eyuyan.com”；未访问时的文本颜色设为绿色；访问后的文本颜色为红色；鼠标移到标题上时标题的热点颜色为黄色；整个超级链接框的背景色设为黑色。

#### 7) “热点颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定鼠标移动到组件上时的文本显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
超级链接框1.标题 = “易语言”  
超级链接框1.类型 = 1  
超级链接框1.Internet地址 = “http://www.eyuyan.com”  
超级链接框1.文本颜色 = #绿色  
超级链接框1.访问后的颜色 = #红色  
超级链接框1.热点颜色 = #黄色  
超级链接框1.背景颜色 = #黑色
```







程序执行后设置“超级链接框1”的标题为“易语言”；链接类型为Internet地址；链接目标为“http://www.eyuyan.com”；未访问时的文本颜色设为绿色；访问后的文本颜色为红色；鼠标移到标题上时标题的热点颜色为黄色；整个超级链接框的背景色设为黑色。

#### 8) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定超级链接框的背景显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
超级链接框1.标题 = “易语言”
超级链接框1.类型 = 1
超级链接框1.Internet地址 = “http://www.eyuyan.com”
超级链接框1.文本颜色 = #绿色
超级链接框1.访问后的颜色 = #红色
超级链接框1.热点颜色 = #黄色
超级链接框1.背景颜色 = #黑色
```

程序执行后设置“超级链接框1”的标题为“易语言”；链接类型为Internet地址；链接目标为“http://www.eyuyan.com”；未访问时的文本颜色设为绿色；访问后的文本颜色为“红色”；鼠标移到标题上时标题的热点颜色为黄色；整个超级链接框的背景色设为黑色。

#### 9) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定超级链接框的显示样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
超级链接框_边框.边框 = 5
```

程序执行后将“超级链接框\_边框”的边框设为单线边框样式。

#### 10) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定超级链接框中标题的字体。

例如：

变量名	类型	静态	数组	备注
变量	字体			

```
超级链接框1.标题 = “易语言”
```

```
变量.字体名称 = “黑体”
```

```
变量.字体大小 = 20
```

```
超级链接框1.字体 = 变量
```





程序执行后将“超级链接框1”的标题设为“易语言”；设置“超级链接框1”的标题应用新的字体。

4.9.4.2 超级链接框命令

“跳转”命令原型为：

<无返回值> 对象. 跳转 ()

本命令用于跳转到属性指定的邮件或 Internet 地址。

例如：

```
超级链接框1.类型 = 1
超级链接框1.Internet地址 = "http://www.eyuyan.com"
超级链接框1.跳转 ()
```

程序执行后，自动跳转到“http://www.eyuyan.com”。

4.10 时间类组件

4.10.1 时钟

4.10.1.1 时钟属性

“时钟周期”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定时钟事件产生的周期，单位为毫秒。如果设定为0，则不产生时钟事件。需要注意的是，如果在程序执行后动态的将本属性设为非0，并不会立即触发“周期事件”。第一次触发事件将在改变属性值后的一个时钟周期（即，若设为100将在100毫秒后触发事件）。

例如：

时钟\_周期.时钟周期 = 1000

程序执行后设置“时钟\_周期”每隔1秒触发一次时钟事件。

4.10.1.2 时钟事件

“周期事件”事件原型为：

无返回值；无参数

每当经过了指定的时钟周期，将触发此事件。

例如：

子程序名	返回值类型	公开	备注
_时钟1_周期事件			

» 输出调试文本 (取现行时间 ())





程序执行后每当“时钟1”的周期事件触发时在输出面板中显示现行时间。

## 4.10.2 月历

### 4.10.2.1 月历属性

#### 1) “不显示今天”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性指定了月历是否在下方显示运行时的现行日期。“真”为不显示，“假”为显示。

例如：

月历\_今天.不显示今天 = 真

程序执行后关闭“月历\_今天”显示现行日期的功能。

#### 2) “不圈注今天”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性指定了月历是否使用红笔效果圈选现行日期和“今天”属性选择日期。“真”为不圈选，“假”为圈选。

例如：

月历\_今天.不圈注今天 = 真

程序执行后关闭“月历\_今天”显示圈选“今天”的效果。

#### 3) “开始星期首日”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定月历列起始星期几。可供选择的属性值有：0（星期一）；1（星期二）；2（星期三）；3（星期四）；4（星期五）；5（星期六）。

例如：

月历\_星期.开始星期首日 = 2

程序执行后设置“月历\_星期”从周三开始排列。

#### 4) “显示星期序号”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定月历是否显示某周是该年的第几周。“真”为显示，“假”为不显示。

例如：

月历\_星期.显示星期序号 = 真

程序执行后设置“月历\_星期”开启显示周序号功能。

#### 5) “滚动月数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定按下月历中的调节按钮时跳过的月数。如本属性为0或1，则默认逐月滚动。

例如：

月历\_月数.滚动月数 = 2

程序执行后设置“月历\_月数”每次滚动2个月。

#### 6) “今天”属性

**属性类型：**日期时间型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定月历现行日期，如果不设定，则自动取系统时间中记录的现行日期。

例如：

月历\_今天.今天 = [2010年1月1日]

程序执行后将“月历\_今天”的现行日期设为“2010年1月1日”。

#### 7) “最小日期”属性

**属性类型：**日期时间型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定能被显示的日期下限。

例如：

月历\_日期.最小日期 = [2000年1月1日]

程序执行后将“月历\_日期”的可调节日期范围下限设为“2000年1月1日”。

#### 8) “最大日期”属性

**属性类型：**日期时间型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定能被显示的日期上限。

例如：

月历\_日期.最大日期 = [2100年1月1日]

程序执行后将“月历\_日期”的可调节日期范围上限设为“2100年1月1日”。

#### 9) “允许选择多天”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否允许在月历中连续选择多个日期。“真”为允许，“假”为不允许。

例如：

月历1.允许选择多天 = 真

月历1.最多选择天数 = 7

月历1.首选择日 = 月历1.今天

月历1.尾选择日 = 增减时间(月历1.今天, #日, 6)

程序执行后，开启“月历1”的连续选择多天功能；设置最多选择7天；选择“今天”及今天后的6天，共7天。





## 10) “最多选择天数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许选择多天”属性为“真”，本属性用于指定最多选择的天数。

例如：

```
月历1.允许选择多天 = 真
月历1.最多选择天数 = 7
月历1.首选择日 = 月历1.今天
月历1.尾选择日 = 增减时间 (月历1.今天, #日, 6)
```

程序执行后，开启“月历1”的连续选择多天功能；设置最多选择7天；选择“今天”及今天后的6天，共7天。

## 11) “首选择日”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许选择多天”属性为“假”本属性用于指定月历现行选择的日期；如果“允许选择多天”属性为“真”，本属性用于指定选择起始日期。

例如：

```
月历1.允许选择多天 = 真
月历1.最多选择天数 = 7
月历1.首选择日 = 月历1.今天
月历1.尾选择日 = 增减时间 (月历1.今天, #日, 6)
```

程序执行后，开启“月历1”的连续选择多天功能；设置最多选择7天；选择“今天”及今天后的6天，共7天。

## 12) “尾选择日”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许选择多天”属性为“真”，本属性用于指定选择到的目标日期。

例如：

```
月历1.允许选择多天 = 真
月历1.最多选择天数 = 7
月历1.首选择日 = 月历1.今天
月历1.尾选择日 = 增减时间 (月历1.今天, #日, 6)
```

程序执行后，开启“月历1”的连续选择多天功能；设置最多选择7天；选择“今天”及今天后的6天，共7天。

## 13) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置月历中文字的字体。





例如：

变量名	类 型	静态	数组	备 注
变量	字体			

变量.字体名称 = “黑体”

变量.字体大小 = 25

月历1.字体 = 变量

程序执行后将“月历1”中的文字应用新的字体。

14) “文本颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定月历中日期文字的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

月历\_颜色.文本颜色 = #绿色

程序执行后设置“月历\_颜色”的日期显示颜色为绿色。

15) “背景颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定月历的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

月历1.文本颜色 = #绿色  
月历1.背景颜色 = #紫色  
月历1.内背景颜色 = #黑色  
月历1.标题颜色 = #黄色  
月历1.标题背景颜色 = #黑色  
月历1.非本月颜色 = #桃红

程序执行后设置“月历1”的日期颜色为绿色；外背景颜色设为紫色；内背景色设为黑色；标题颜色设为黄色；标题背景色设为黑色；非本月日期颜色设为桃红。

16) “内背景颜色” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定用来实际显示月历部分的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

月历1.文本颜色 = #绿色  
月历1.背景颜色 = #紫色  
月历1.内背景颜色 = #黑色





```
月历1.标题颜色 = #黄色  
月历1.标题背景颜色 = #黑色  
月历1.非本月颜色 = #桃红
```

程序执行后设置“月历1”的日期颜色为绿色；外背景颜色设为紫色；内背景色设为黑色；标题颜色设为黄色；标题背景色设为黑色；非本月日期颜色设为桃红。

#### 17) “标题颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定月历标题文字的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
月历1.文本颜色 = #绿色  
月历1.背景颜色 = #紫色  
月历1.内背景颜色 = #黑色  
月历1.标题颜色 = #黄色  
月历1.标题背景颜色 = #黑色  
月历1.非本月颜色 = #桃红
```

程序执行后设置“月历1”的日期颜色为绿色；外背景颜色设为紫色；内背景色设为黑色；标题颜色设为黄色；标题背景色设为黑色；非本月日期颜色设为桃红。

#### 18) “标题背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定月历标题的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
月历1.文本颜色 = #绿色  
月历1.背景颜色 = #紫色  
月历1.内背景颜色 = #黑色  
月历1.标题颜色 = #黄色  
月历1.标题背景颜色 = #黑色  
月历1.非本月颜色 = #桃红
```

程序执行后设置“月历1”的日期颜色为绿色；外背景颜色设为紫色；内背景色设为黑色；标题颜色设为黄色；标题背景色设为黑色；非本月日期颜色设为桃红。

#### 19) “非本月颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定月历中不属于本月的日期显示颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。





例如：

月历1.文本颜色 = #绿色  
月历1.背景颜色 = #紫色  
月历1.内背景颜色 = #黑色  
月历1.标题颜色 = #黄色  
月历1.标题背景颜色 = #黑色  
月历1.非本月颜色 = #桃红

程序执行后设置“月历1”的日期颜色为绿色；外背景颜色设为紫色；内背景色设为黑色；标题颜色设为黄色；标题背景色设为黑色；非本月日期颜色设为桃红。

20) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定月历边框的显示样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

月历\_边框.边框 = 5

程序执行后设置“月历\_边框”的边框样式为单线边框式。

4.10.2.2 月历事件

“选择日期被改变”事件原型为：

**返回值类型：**无返回值；**无参数**

当现行被选择的日期或区域被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备注
_月历1_选择日期被改变			

月历1.今天 = 月历1.首选择日

程序执行后，当“月历1”中选择的日期被改变后，将“今天”设为选择的日期。

4.10.3 日期框

4.10.3.1 日期框属性

1) “允许编辑”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否允许在运行时手动输入日期。“真”为允许，“假”为不允许。

例如：

日期框\_属性.允许编辑 = 真

程序执行后将“日期框\_属性”的允许手动输入日期功能开启。





## 2) “附件类型” 属性

**属性类型:** 整数型; **有效范围:** 设计时、编程时; **编程时权限:** 使用、读取、更改  
本属性用于指定半自动调节日期时的方式。可供选择的属性值有: 0 (下拉月历);

1 (调节器)。

例如:

日期框\_属性.附件类型 = 1

程序执行后将“日期框\_属性”的调节方式设为调节器样式。

## 3) “今天” 属性

**属性类型:** 日期时间型; **有效范围:** 设计时、编程时; **编程时权限:** 使用、读取、更改  
本属性用于指定或读取现行日期, 如果不设定, 则自动取现行日期。

例如:

日期框\_属性.今天 = [2010年1月1日]

程序执行后将“日期框\_属性”的当前显示日期设为“2010年1月1日”。

## 4) “最小日期” 属性

**属性类型:** 日期时间型; **有效范围:** 设计时、编程时; **编程时权限:** 使用、读取、更改  
本属性用于设置日期框调节日期范围的下限值。

例如:

日期框\_属性.最小日期 = [2000年1月1日]

程序执行后设置“日期框\_属性”的调节范围下限值为“2000年1月1日”。

## 5) “最大日期” 属性

**属性类型:** 日期时间型; **有效范围:** 设计时、编程时; **编程时权限:** 使用、读取、更改  
本属性用于设置日期框调节日期范围的上限值。

例如:

日期框\_属性.最大日期 = [2100年1月1日]

程序执行后设置“日期框\_属性”的调节范围上限值为“2100年1月1日”。

## 6) “字体” 属性

**属性类型:** 字体; **有效范围:** 设计时、编程时; **编程时权限:** 使用、读取、更改  
本属性用于设置日期框和其附件框中的文字字体。

例如:

变量名	类型	静态	数组	备注
变量	字体			

变量.字体名称 = “黑体”

变量.字体大小 = 25

日期框1.字体 = 变量





程序执行后将“日期框1”的当前字体设为新的字体。

7) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置日期框的边框样式。可供选择的属性值有：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

日期框\_属性.边框 = 5

程序执行后将设置“日期框\_属性”边框样式为单线边框式。

8) “数据源”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于与指定提供数据的数据源组件绑定。

例如：

日期框1.数据源 = “数据源1”  
日期框1.数据列 = “date”

程序执行后将“日期框1”的内容与“数据源1”中当前记录的“date”字段的值绑定。

9) “数据列”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于与指定数据源中的数据列绑定，可以是列名或以1开始的列序号文本。

例如：

日期框1.数据源 = “数据源1”  
日期框1.数据列 = “date”

程序执行后将“日期框1”的内容与“数据源1”中当前记录的“date”字段的值绑定。

4.10.3.2 日期框事件

“选择日期被改变”事件原型为：

**返回值类型：**无返回值

当现行被选择的日期被改变后触发本事件。

例如：

子程序名	返回值类型	公开	备注
_日期框1_选择日期被改变			

» 输出调试文本（日期框1.今天）

程序运行后，当“日期框1”的现行时间被改变后，显示被改变后的时间。





## 4.11 显示类组件

### 4.11.1 编辑框

#### 4.11.1.1 编辑框属性

##### 1) “内容”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框中保存和显示的内容。

例如：

```
编辑框_属性.内容 = “易语言”
```

程序执行后将“编辑框\_属性”的内容改变为“易语言”。

##### 2) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框边框的样式。

样式值可以是：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）。

例如：

```
编辑框_属性.边框 = 5
```

程序执行后将“编辑框\_属性”的边框改为单线边框。

##### 3) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框内文本显示的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
编辑框_属性.文本颜色 = #蓝色
```

程序执行后会将“编辑框\_属性”的文本颜色设为蓝色。

##### 4) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框内的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
编辑框_属性.背景颜色 = #红色
```





程序执行后会将“编辑框\_属性”的背景颜色设为红色。

5) “字体” 属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框内文字所使用的字体。

例如：

变量名	类 型	静态	数组	备 注
变量1	字体			

变量1.加粗 = 真  
变量1.下划线 = 真  
变量1.字体名称 = “黑体”  
变量1.字体大小 = 30  
编辑框1.字体 = 变量1

程序执行后会为“编辑框1”中的文字设置一个新的字体。

6) “隐藏选择” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性指定失去输入焦点后是否还显示当前被选择的区域。“真”为显示，“假”为不显示。

例如：

编辑框\_属性.隐藏选择 = 假

程序执行后会关闭失去焦点时显示选中内容的功能。

7) “最大允许长度” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置最大允许输入到本组件中的字符数目，如为0则输入字符数目不受限制。

例如：

编辑框\_属性.最大允许长度 = 10

程序执行后将“编辑框\_属性”内最大允许存在字符数设为10。

8) “允许多行” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置是否允许输入多行文本。为“真”表示允许，“假”为不允许。  
如果为“真”，将会根据“滚动条”属性的类型自动换行（加换行符及回车换行）；为“假”，则忽略换行符和回车，在一行内显示所有文本。

例如：

编辑框1.是否允许多行 = 真  
编辑框1.滚动条 = 2





程序执行后将“编辑框1”设为允许多行，并添加纵向滚动条。

#### 9) “滚动条”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“允许多行”为“真”，本属性用于设置编辑框是否显示滚动条。

本属性可以是以下值之一：0（无）；1（横向滚动条）；2（纵向滚动条）；3（横向及纵向滚动条）。

例如：

```
编辑框1.是否允许多行 = 真
编辑框1.滚动条 = 2
```

程序执行后将“编辑框1”设为允许多行，并添加纵向滚动条。

#### 10) “对齐方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定编辑框中内容的对齐方式。

对齐方式可以是以下值之一：0（左对齐）；1（居中）；2（右对齐）。

例如：

```
编辑框_属性.对齐方式 = 1
```

程序执行后将“编辑框\_属性”的对齐方式改变为居中对齐。

#### 11) “输入方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置用户在编辑框内的输入方式。

输入方式可以是以下值之一：0（通常方式）；1（只读方式）；2（密码输入）；3（整数文本输入）；4（小数文本输入）；5（输入字节）；6（输入短整数）；7（输入整数）；8（输入长整数）；9（输入小数）；10（输入双精度小数）；11（输入日期时间）。

例如：

```
编辑框_属性.输入方式 = 1
```

程序执行后将“编辑框\_属性”输入方式设为只读。

#### 12) “密码掩盖字符”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置当输入方式为“密码输入”时输入字符后显示在输入框中的替代字符。

例如：

```
编辑框1.输入方式 = 2
编辑框1.密码掩盖字符 = “#”
```

程序执行后将“编辑框1”的输入方式设为“密码输入”并指定输入的内容使用“#”





替换显示。

### 13) “转换方式” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置是否使用自动大小写转换功能。

本属性可以是以下值之一：0（无）；1（大写转为小写）；2（小写转为大写）。

例如：

```
编辑框_属性.转换方式 = 1
```

程序执行后开启“编辑框\_属性”的大写自动转换为小写功能。

### 14) “调节器方式” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置编辑框调节器方式。调节器只有在单行方式下才有效。

本属性可以是以下值之一：0（无）；1（自动调节）；2（手动调节）。当为手动调节时按下调节器会触发编辑框的“调节钮被按下”事件。

例如：

```
编辑框_属性.调节器方式 = 1
```

程序执行后会将“编辑框\_属性”调节器设为自动式调节器。

### 15) “调节器底限值” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果编辑框“调节器方式”属性为自动调节器方式时，本属性用于设置自动调节范围的最小值。最小不能小于-32767。

例如：

```
编辑框1.调节器方式 = 1  
编辑框1.调节器底限值 = -100  
编辑框1.调节器上限值 = 100
```

程序执行后设置“编辑框1”显示自动式调节器；设置调节器最小值为“-100”；设置调节器上限为“100”。

### 16) “调节器上限值” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果编辑框“调节器方式”属性为自动调节器方式时，本属性用于设置自动调节范围的最大值。最大不能超过32767。

例如：

```
编辑框1.调节器方式 = 1  
编辑框1.调节器底限值 = -100  
编辑框1.调节器上限值 = 100
```

程序执行后设置“编辑框1”显示自动式调节器；设置调节器最小值为“-100”；设





置调节器上限为“100”。

#### 17) “起始选择位置”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于获取或设置编辑框内所选择文本的起始点；0 为位置 1，1 为位置 2，以此类推。如果没有文本被选中，则指出光标位置。如果设置位置时使用值 -1，则将当前光标位置移动到文本尾部。

例如：

```
编辑框1.起始选择位置 = 3
编辑框1.被选择字符数 = 5
» 输出调试文本 (编辑框1.被选择文本)
编辑框1.被选择文本 = “123”
```

程序执行后将起始选择位置指定为第3个字后；选择5个字节文字；将选中的文字显示在输出面板；将被选中的内容替换成“123”。

#### 18) “被选择字符数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于获取或设置编辑框内所选择的字符数；如果设置字符数时使用值 -1，则选择编辑框内的所有字符。

例如：

```
编辑框1.起始选择位置 = 3
编辑框1.被选择字符数 = 5
» 输出调试文本 (编辑框1.被选择文本)
编辑框1.被选择文本 = “123”
```

程序执行后将起始选择位置指定为第3个字后；选择5个字节文字；将选中的文字显示在输出面板；将被选中的内容替换成“123”。

#### 19) “被选择文本”属性

**属性类型：**文本型；**有效范围：**编程时；**编程时权限：**使用、读取、更改

本属性用于获取或替换当前所选择的文本。

例如：

```
编辑框1.起始选择位置 = 3
编辑框1.被选择字符数 = 5
» 输出调试文本 (编辑框1.被选择文本)
编辑框1.被选择文本 = “123”
```

程序执行后将起始选择位置指定为第3个字后；选择5个字节文字；将选中的文字显示在输出面板；将被选中的内容替换成“123”。





20) “数据源” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

```
标签1.数据源 = “数据源1”  
标签1.数据列 = “name”
```

程序执行后将“编辑框1”的数据来源和“数据源1”绑定；将编辑框显示内容与数据源当前记录中“name”列绑定。

21) “数据列” 属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```
编辑框1.数据源 = “数据源1”  
编辑框1.数据列 = “name”
```

程序执行后将“编辑框1”的数据来源和“数据源1”绑定；将编辑框显示内容与数据源当前记录中“name”列绑定。

4.11.1.2 编辑框事件

1) “内容被改变” 事件

事件原型为：

无返回值；无参数

当编辑框的内容被修改后即产生此事件。

例如：

子程序名	返回值类型	公开	备注
_编辑框1_内容被改变			

» 输出调试文本 (编辑框1.内容)

程序执行后，当“编辑框1”中的内容改变时，在输出面板中显示当前“编辑框1”中的内容。

2) “调节钮被按下” 事件

事件原型为：

返回值类型：无返回值

参数一：参数名：按钮值 类型：整数型



当编辑框的“调节器方式”为“手动调节器”，调节按钮被按下时触发本事件。

参数<1>用于判断具体按下的是调节器中的哪个按钮。如果按下的是调节器的向上箭头按钮，参数值为 1，否则为 -1。

例如：

子程序名	返回值类型	公开	备注		
编辑框1_调节钮被按下					
参数名	类型	参考	可空	数组	备注
按钮值	整数型				

» 输出调试文本 (选择 (按钮值 > 0, “向上”, “向下”))

程序执行后，当“编辑框1”的调节器被按下时会在输出面板中显示对应的文字。

4.11.1.3 编辑框命令

“加入文本”命令原型为：

<无返回值> 对象. 加入文本 (文本型 欲加入文本, ...)

本命令用于将指定文本加入到编辑框内容的尾部。命令参数表中最后一个参数可以被重复添加。

例如：

编辑框\_命令.加入文本 (“易”, “语言”)

程序执行后向“编辑框\_命令”内依次添加“易”、“语言”。

4.11.2 标签

4.11.2.1 标签属性

1) “标题”属性

属性类型：文本型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于指定标签所显示的标题文字。如果希望某一字符成为本标签后可停留焦点组件的访问键，可以在该字符前面加上一个“&”字符。

例如：

标签\_属性.标题 = “易语言”

程序执行后将“标签\_属性”的标题设为“易语言”。

2) “效果”属性

属性类型：整数型；有效范围：设计时、编程时；编程时权限：使用、读取、更改

本属性用于指定标签所显示文字的显示效果。

属性值可以是以下值之一：0（通常）；1（凹入）；2（凸出）；3（阴影）。



例如：

```
标签_属性.标题 = “易语言”
```

```
标签_属性.效果 = 3
```

程序执行后将“标签\_属性”的标题设为“易语言”；将“标签\_属性”标题显示效果设为阴影样式。

### 3) “边框”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定标签边框显示效果。

属性值可以是以下值之一：0（无边框）；1（凹入式）；2（凸出式）；3（浅凹入式）；4（镜框式）；5（单线边框式）；6（渐变镜框式）。

例如：

```
标签1.边框 = 6
```

```
标签1.渐变边框宽度 = 10
```

```
标签1.渐变边框颜色1 = #红色
```

```
标签1.渐变边框颜色2 = #绿色
```

```
标签1.渐变边框颜色3 = #蓝色
```

程序执行后将“标签1”的边框设为渐变镜框式；设置渐变框宽度为10像素；设置起始渐变色为红色；设置过度渐变色为绿色；设置最终渐变色为蓝色。

### 4) “渐变边框宽度”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“边框”属性值为“6（渐变镜框式）”时，本属性用于指定渐变边框的宽度，单位为像素值。

例如：

```
标签1.边框 = 6
```

```
标签1.渐变边框宽度 = 10
```

```
标签1.渐变边框颜色1 = #红色
```

```
标签1.渐变边框颜色2 = #绿色
```

```
标签1.渐变边框颜色3 = #蓝色
```

程序执行后将“标签1”的边框设为渐变镜框式；设置渐变框宽度为10像素；设置起始渐变色为红色；设置过度渐变色为绿色；设置最终渐变色为蓝色。

### 5) “渐变边框颜色1”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“边框”属性值为“6（渐变镜框式）”时，本属性用于设置标签边框的起始渐变色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。





例如：

```
标签1.边框 = 6
标签1.渐变边框宽度 = 10
标签1.渐变边框颜色1 = #红色
标签1.渐变边框颜色2 = #绿色
标签1.渐变边框颜色3 = #蓝色
```

程序执行后将“标签1”的边框设为渐变镜框式；设置渐变框宽度为10像素；设置起始渐变色为红色；设置过度渐变色为绿色；设置最终渐变色为蓝色。

#### 6) “渐变边框颜色2”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“边框”属性值为6时，本属性用于设置标签边框的过度渐变色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
标签1.边框 = 6
标签1.渐变边框宽度 = 10
标签1.渐变边框颜色1 = #红色
标签1.渐变边框颜色2 = #绿色
标签1.渐变边框颜色3 = #蓝色
```

程序执行后将“标签1”的边框设为渐变镜框式；设置渐变框宽度为10像素；设置起始渐变色为红色；设置过度渐变色为绿色；设置最终渐变色为蓝色。

#### 7) “渐变边框颜色3”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果“边框”属性值为“6（渐变镜框式）”时，本属性用于设置标签边框的最终渐变色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
标签1.边框 = 6
标签1.渐变边框宽度 = 10
标签1.渐变边框颜色1 = #红色
标签1.渐变边框颜色2 = #绿色
标签1.渐变边框颜色3 = #蓝色
```

程序执行后将“标签1”的边框设为渐变镜框式；设置渐变框宽度为10像素；设置起始渐变色为红色；设置过度渐变色为绿色；设置最终渐变色为蓝色。

#### 8) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于设置标签内文字所使用的字体。

例如：

变量名	类型	静态	数组	备注
变量1	字体			

变量1.加粗 = 真

变量1.下划线 = 真

变量1.字体名称 = “黑体”

变量1.字体大小 = 30

标签1.字体 = 变量1

程序执行后会为“标签1”中的文字设置一个新的字体。

#### 9) “文本颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置标签内标题的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

标签\_属性.文本颜色 = #蓝色

程序执行后将“标签\_属性”的标题颜色设为蓝色。

#### 10) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置标签内背景的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

标签\_属性.背景颜色 = #黄色

程序执行后将“标签\_属性”的背景颜色设为黄色。

#### 11) “底图”属性

**属性类型：**字节集；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性指定显示在标签背景上的图片。

例如：

标签1.底图 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
标签1.底图方式 = 1

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“标签1”的底图；将底图显示方式设为平铺。

#### 12) “底图方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





如果设定了底图，本属性用于指定标签背景上底图的显示方式。

显示方式有：0（图片居上）；1（图片平铺）；2（图片居中）；3（缩放图片）。

例如：

```
标签1.底图 = 读入文件 ("C:\WINDOWS\Prairie Wind.bmp")
标签1.底图方式 = 1
```

程序执行后会读入“C:\WINDOWS\Prairie Wind.bmp”（假设文件存在）作为“标签1”的底图；将底图显示方式设为平铺。

### 13) “渐变背景方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定标签背景色渐变样式。本属性只有在未设定底图的前提下才有效。

本属性可以是以下值之一：0（无渐变背景）；1（从上到下）；2（从左到右）；3（从左上到右下）；4（从右上到左下）；5（从下到上）；6（从右到左）；7（从右下到左上）；8（从左下到右上）。

例如：

```
标签1.渐变背景方式 = 3
标签1.渐变背景颜色1 = #紫色
标签1.渐变背景颜色2 = #嫩绿
标签1.渐变背景颜色3 = #深青
```

程序执行后将“标签1”的渐变背景样式设为从左上到右下；设置起始渐变色为紫色；设置过度渐变色为嫩绿；设置最终渐变色为深青。

### 14) “渐变背景颜色1”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

当“渐变背景方式”属性不为0时，本属性用于设置标签背景颜色渐变时的起始颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
标签1.渐变背景方式 = 3
标签1.渐变背景颜色1 = #紫色
标签1.渐变背景颜色2 = #嫩绿
标签1.渐变背景颜色3 = #深青
```

程序执行后将“标签1”的渐变背景样式设为从左上到右下；设置起始渐变色为紫色；设置过度渐变色为嫩绿；设置最终渐变色为深青。

### 15) “渐变背景颜色2”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





当“渐变背景方式”属性不为0时，本属性用于设置标签背景颜色渐变时的过度颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
标签1.渐变背景方式 = 3  
标签1.渐变背景颜色1 = #紫色  
标签1.渐变背景颜色2 = #嫩绿  
标签1.渐变背景颜色3 = #深青
```

程序执行后将“标签1”的渐变背景样式设为从左上到右下；设置起始渐变色为紫色；设置过度渐变色为嫩绿；设置最终渐变色为深青。

#### 16) “渐变背景颜色3”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

当“渐变背景方式”属性不为0时，本属性用于设置标签背景颜色渐变时的最终颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
标签1.渐变背景方式 = 3  
标签1.渐变背景颜色1 = #紫色  
标签1.渐变背景颜色2 = #嫩绿  
标签1.渐变背景颜色3 = #深青
```

程序执行后将“标签1”的渐变背景样式设为从左上到右下；设置起始渐变色为紫色；设置过度渐变色为嫩绿；设置最终渐变色为深青。

#### 17) “横向对齐方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置标签内文字的横向对齐方式。

本属性可以是以下值之一：0（左对齐）；1（居中）；2（右对齐）。

例如：

```
标签_属性.横向对齐方式 = 1
```

程序执行后将“标签\_属性”的内容横向对齐方式设为居中对齐。

#### 18) “是否自动折行”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设定当文本内容超出右界后是否自动转到下一行显示。

例如：

```
标签_属性.是否自动折行 = 真
```

程序执行后将“标签\_属性”的自动折行功能开启。





## 19) “纵向对齐方式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
当“是否自动折行”属性为“假”时，本属性用于设置标签内文字的纵向对齐方式。  
本属性可以是以下值之一：0（顶对齐）；1（居中）；2（底对齐）。

例如：

```
标签_属性.纵向对齐方式 = 2
```

程序执行后将“标签\_属性”的内容纵向对齐方式设为底对齐。

## 20) “数据源”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定提供数据的数据源组件名。与数据源绑定后会自动显示当前记录中由“数据列”属性所指定列的数据。

例如：

```
标签1.数据源 = “数据源1”  
标签1.数据列 = “name”
```

程序执行后将“标签1”的数据来源和“数据源1”绑定；将标签显示内容与数据源当前记录中“name”列绑定。

## 21) “数据列”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
如果指定了“数据源”属性，本属性用于指定数据源中的数据列，可以是列名或以1开始的列序号文本。

例如：

```
标签1.数据源 = “数据源1”  
标签1.数据列 = “name”
```

程序执行后将“标签1”的数据来源和“数据源1”绑定；将标签显示内容与数据源当前记录中“name”列绑定。

## 4.11.2.2 标签事件

“反馈事件”原型为：

返回值类型：整数型

参数一：参数名：参数一 类型：整数型

参数二：参数名：参数二 类型：整数型

本事件用于与外部DLL交互或者其他场合，例如在使用“同步播放MP3”命令时作为播放进度，事件子程序处理完毕后可以根据需要决定是否返回值（信息值为32885）。

参数<1>保存了触发事件时获得的第一个参数。

参数<2>保存了触发事件时获取的第二个参数。





例如：

子程序名	返回值类型	公开	备 注		
_标签1_反馈事件	整数型				
参数名	类 型	参考	可空	数组	备 注
参数一	整数型				
参数二	整数型				

返回 (参数一 × 参数二)

子程序名	返回值类型	公开	备 注
_启动窗口_创建完毕			

变量名	类 型	静态	数组	备 注
结果	整数型			

结果 = 标签1.调用反馈事件 (5, 5, 真)

» 输出调试文本 (结果)

“标签1”的反馈事件中将获得的两个参数相乘并将计算结果返回；程序执行后，在“\_启动窗口”创建完毕时调用“标签1”的反馈事件，并将返回的结果显示在输出面板中。

本例程源码见随书光盘中“\图书例程\第四章\标签\反馈事件.e”

### 4.11.2.3 标签命令

“调用反馈事件”命令原型为：

<整数型> 对象. 调用反馈事件 ( [整数型 参数一] , [整数型 参数二] , [逻辑型 事件传递方式] )

本命令用于产生标签的反馈事件，以调用此标签的“反馈事件”用户事件处理子程序，可以用作在多线程处理中将控制权转移到程序主线程上去执行。返回用户事件处理子程序所返回的值，如果没有相应的事件处理子程序或采用投递方式传递事件消息，将返回零。

参数<1>指定了向事件中传递的第一个数据的值。

参数<2>指定了向事件中传递的第二个数据的值。

参数<3>指定了传递事件的方式。为真则采用发送方式传递事件消息，此时本命令将一直等待到“反馈事件”，用户事件处理子程序，处理完毕后才返回；为假则采用投递方式传递事件消息，此时本命令将直接返回且不等待用户事件处理子程序处理完毕（用户处理子程序将在空闲时被调用）。如果本参数被省略，默认采用发送方式传递事件。

例如：

子程序名	返回值类型	公开	备 注		
_标签1_反馈事件	整数型				
参数名	类 型	参考	可空	数组	备 注
参数一	整数型				
参数二	整数型				

返回 (参数一 × 参数二)





子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
结果	整数型			

结果 = 标签1.调用反馈事件 (5, 5, 真)

» 输出调试文本 (结果)

“标签1”的反馈事件中将获得的两个参数相乘并将计算结果返回；程序执行后，在“\_启动窗口”创建完毕时调用“标签1”的反馈事件，并将返回的结果显示在输出面板中。

本例程源码见随书光盘中“\图书例程\第四章\标签\反馈事件.e”

### 4.11.3 表格

表格一般用于显示一组或多组有固定格式的数据。例如显示数据库中的内容等。由于表格只提供基本显示操作的功能。其中的数据，及内容格式来自数据源，所以表格一般和数据源配合使用。

#### 4.11.3.1 表格属性

##### 1) “数据源”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定提供数据的数据源组件名。

例如：

表格\_属性.数据源 = “数据源1”

程序执行后将“表格\_属性”与“数据源1”绑定。

##### 2) “表格线颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定表格中表格线的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

表格\_属性.表格线颜色 = #绿色

程序执行后将“表格\_属性”的表格线颜色设为绿色。

##### 3) “背景颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定表格的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

表格\_属性.背景颜色 = #黑色

程序执行后将“表格\_属性”的背景颜色设为黑色。





## 4) “缩放比” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定显示表格时所使用的缩放百分比，值在 20 到 1000 之间。

例如：

表格\_属性.缩放比 = 50

程序执行后将“表格\_属性”的缩放比设为50%。

## 5) “允许选择块” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否允许操作者选择表格单元格区域。“真”为允许，“假”为不允许。

例如：

表格\_属性.允许选择块 = 假

程序执行后设置“表格\_属性”不允许选择表格中一定的单元格区域。

## 6) “显示标尺” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定表格是否显示左边和顶边的标尺。“真”为显示，“假”为不显示。

例如：

表格\_属性.显示标尺 = 假

程序执行后设置“表格\_属性”中不显示标尺。

## 7) “显示空格线” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定是否用虚线显示空表格线。“真”为显示，“假”为不显示。

例如：

表格\_属性.显示空表格线 = 假

程序执行后设置“表格\_属性”不显示未选中单元格的表格线。

## 8) “禁止调整行高” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本命令用于指定是否禁止操作者调整表格行高。“真”为禁止调整，“假”为允许调整。

例如：

表格\_属性.禁止调整行高 = 真

程序执行后设置“表格\_属性”不允许在运行时手动调整表格的行高。

## 9) “禁止调整列宽” 属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本命令用于指定是否禁止操作者调整表格列宽。“真”为禁止调整，“假”为允许调整。

例如：

表格\_属性.禁止调整列宽 = 真

程序执行后设置“表格\_属性”不允许在运行时手动调整表格的列宽。

10) “允许粘贴扩展”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本命令用于指定当粘贴表格数据时，如果现行表格尺寸无法容纳，是否允许自动扩展。“真”为允许，“假”为不允许。

例如：

表格\_属性.允许粘贴扩展 = 假

程序执行后将“表格\_属性”不允许自动扩展。

#### 4.11.3.2 表格事件

1) “光标位置改变”事件

事件原型为：

**返回值类型：**无返回值；无参数

当操作者改变了光标位置后会产生本事件。

例如：

子程序名	返回值类型	公开	备注
_表格1_光标位置改变			

» 输出调试文本（“光标位置被改变”）

程序执行后，当“表格1”中的光标位置被用户改变时，在输出面板中显示一段文字。

2) “选择行列数改变”事件

事件原型为：

**返回值类型：**无返回值；无参数

当操作者更改了当前被选择区域的行列数时会产生本事件。

例如：

子程序名	返回值类型	公开	备注
_表格1_选择行列数改变			

» 输出调试文本（表格1.取选择行数 0）

» 输出调试文本（表格1.取选择列数 0）

程序执行后，当“表格1”被选择的行数或列数被改变时，在输出面板中显示当前被选择的行数和列数。





3) “内容被改变”事件

事件原型为：

返回值类型：无返回值

参数一：参数名：行号 类型：整数型

参数二：参数名：列号 类型：整数型

参数三：参数名：行数 类型：整数型

参数四：参数名：列数 类型：整数型

当操作者修改了单元格内容时即产生此事件。

参数<1>为触发事件时的起始行号。

参数<2>为触发事件时的起始列号。

参数<3>为被改变的行数。

参数<4>为被改变的列数。

例如：

子程序名	返回值类型	公开	备 注		
_表格1_内容被改变					
参数名	类 型	参 考	可 空	数 组	备 注
行号	整数型				
列号	整数型				
行数	整数型				
列数	整数型				

» 输出调试文本（“内容被改变”）

程序执行后，当“表格1”中单元格的内容被改变时，在输出面板中显示一段文字。

4) “行高被改变”事件

事件原型为：

返回值类型：无返回值

参数一：参数名：行号 类型：整数型

当操作者调整了表格行行高时即产生此事件。

例如：

子程序名	返回值类型	公开	备 注		
_表格1_行高被改变					
参数名	类 型	参 考	可 空	数 组	备 注
行号	整数型				

» 输出调试文本（行号）

程序执行后，当“表格1”的某一行行高被改变时，在输出面板中显示被改变高度的行号。





## 5) “列宽被改变”事件

事件原型为:

返回值类型: 无返回值

参数一: 参数名: 行号 类型: 整数型

当操作者调整了表格列列宽时即产生此事件。

例如:

子程序名	返回值类型	公开	备 注		
_表格1_列宽被改变					
参数名	类 型	参 考	可 空	数 组	备 注
列号	整数型				

» 输出调试文本 (列号)

程序执行后, 当“表格1”的某一列列宽被改变时, 在输出面板中显示被改变列宽的列号。

## 6) “尺寸被扩展”事件

事件原型为:

返回值类型: 无返回值; 无参数

当操作者粘贴来自剪贴板的表格单元数据时, 如果为了容纳该数据自动扩展了表格尺寸, 即产生此事件。

例如:

子程序名	返回值类型	公开	备 注
_表格1_尺寸被扩展			

» 输出调试文本 (“尺寸被扩展”)

程序执行后, 当“表格1”尺寸被扩展时, 在输出面板中显示一段文字。

## 4.11.3.3 表格命令

## 1) “置光标”命令

命令原型为:

<无返回值> 对象. 置光标 (整数型 行号, 整数型 列号)

本命令用于改变表格中当前光标所在位置。

参数<1>指定了目标所在的行号。行号从1开始。

参数<2>指定了目标所在的列号。列号从1开始。

例如:

表格\_命令.置光标 (2, 1)

程序执行后, 将“表格\_命令”中的光标移动到第二行第一个列。





## 2) “选择”命令

命令原型为:

<无返回值> 对象.选择 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数])

本命令用于选择表格中指定范围的单元格。

参数<1>指定了欲选择的区域起始行号。行号从1开始。

参数<2>指定了欲选择区域的起始列号。列号从1开始。

参数<3>指定了欲选择的行数。如果本参数被省略，默认值为1。

参数<4>指定了欲选择的列数。如果本参数被省略，默认值为1。

例如:

表格\_命令.选择 (1, 1, 2, 2)

程序执行后会选中“表格1”中的第一行第一列开始的两行两列。

## 3) “全部选择”命令

命令原型为:

<无返回值> 对象.全部选择 ()

本命令用于选中表格中的所有单元格。

例如:

表格\_命令.全部选择 ()

程序执行后会选中“表格\_命令”中的所有单元格。

## 4) “取光标行号”命令

命令原型为:

<整数型> 对象.取光标行号 ()

本命令用于获取当前光标所在行号。

例如:

输出调试文本 (表格\_命令.取光标行号 ())

输出调试文本 (表格\_命令.取光标列号 ())

程序执行后会在输出面板中显示当前光标所在的行号及列号。

## 5) “取光标列号”命令

命令原型为:

<整数型> 对象.取光标列号 ()

例如:

输出调试文本 (表格\_命令.取光标行号 ())

输出调试文本 (表格\_命令.取光标列号 ())

程序执行后会在输出面板中显示当前光标所在的行号及列号。





## 6) “取选择行数” 命令

命令原型为:

&lt;整数型&gt; 对象. 取选择行数 ()

本命令用于获取被选择的区域中有多少行。

例如:

子程序名	返回值类型	公开	备注
_表格1_选择行列数改变			

» 输出调试文本 (表格1. 取选择行数 0)

» 输出调试文本 (表格1. 取选择列数 0)

程序执行后, 当“表格1”被选择的行数或列数被改变时, 在输出面板中显示当前被选择的行数和列数。

## 7) “取选择列数” 命令

命令原型为:

&lt;整数型&gt; 对象. 取选择列数 ()

本命令用于获取被选择的区域中有多少列。

例如:

子程序名	返回值类型	公开	备注
_表格1_选择行列数改变			

» 输出调试文本 (表格1. 取选择行数 0)

» 输出调试文本 (表格1. 取选择列数 0)

程序执行后, 当“表格1”被选择的行数或列数被改变时, 在输出面板中显示当前被选择的行数和列数。

## 8) “等宽缩放” 命令

命令原型为:

&lt;无返回值&gt; 对象. 等宽缩放 ()

本命令用于将表格内容缩放到与表格窗口等宽显示。

例如:

表格\_命令. 等宽缩放 ()

程序执行后, 使“表格\_命令”中的内容缩放到与表格窗口等宽。

## 9) “全部复制” 命令

命令原型为:

&lt;无返回值&gt; 对象. 全部复制 ()





本命令用于复制所有表格数据到剪贴板。

例如：

表格\_命令.全部复制()

表格\_命令.粘贴到光标处()

程序执行后复制“表格\_命令”中的所有内容并粘贴到当前光标处。

#### 10) “粘贴到光标处”命令

命令原型为：

<无返回值> 对象. 粘贴到光标处 ()

本命令用于粘贴剪贴板中的表格数据到当前光标处。

例如：

表格\_命令.全部复制()

表格\_命令.粘贴到光标处()

程序执行后复制“表格\_命令”中的所有内容并粘贴到当前光标处。

#### 11) “复制”命令

命令原型为：

<无返回值> 对象. 复制 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数])

本命令用于复制指定区域表格数据到剪贴板。

参数<1>指定了欲复制区域的起始行号。行号从1开始。

参数<2>指定了欲复制区域的起始列号。列号从1开始。

参数<3>指定了欲复制的行数。如果本参数被省略，默认值为1。

参数<4>指定了欲复制的列数。如果本参数被省略，默认值为1。

例如：

表格\_命令.复制(1, 1, 2, 2)

表格\_命令.粘贴(3, 3)

程序执行后将“表格\_命令”中第一行第一列、第二列，第二行第一列、第二列中的内容复制到第三行第三列后面。

#### 12) “粘贴”命令

命令原型为：

<无返回值> 对象. 粘贴 (整数型 行号, 整数型 列号)

本命令用于粘贴剪贴板中的表格数据到指定位置。

参数<1>指定了粘贴目标的起始行号。行号从1开始。

参数<2>指定了粘贴目标的起始列号。列号从1开始。







例如：

表格\_命令.复制 (1, 1, 2, 2)

表格\_命令.粘贴 (3, 3)

程序执行后将“表格\_命令”中第一行第一列、第二列，第二行第一列、第二列中的内容复制到第三行第三列后面。

### 13) “打印”命令

命令原型为：

<文本型> 对象.打印 ([逻辑型 是否显示进度框], [文本型 打印作业标题])

本命令用于将表格中的数据打印。注意欲设置打印份数等打印设置信息请到与表格相连接的数据源中设置。成功返回空文本，失败（不包含操作者中断打印）返回非空错误文本。

参数<1>指定了在打印的同时是否显示打印进度框，操作者可以在进度框内中止打印。“真”为显示进度框，“假”为不显示。如果本参数被省略，默认值为“真”。

参数<2>指定可打印时显示在打印机状态窗口中的文档名称。如果参数被省略，默认值为“报表”。

例如：

表格\_命令.打印 (真, “打印表格”)

程序执行后将“表格\_命令”中的数据打印。

### 14) “打印预览”命令

命令原型为：

<无返回值> 对象.打印预览 ()

本命令用于在打印前预览打印效果。

例如：

表格\_命令.打印预览 ()

程序执行后预览“表格\_命令”中数据的打印效果。

## 4.11.4 打印机

### 4.11.4.1 打印机属性

#### 1) “绘画单位”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定绘画时所使用的单位，坐标系的 X 轴从左到右，Y 轴从上到下。本属性可以是以下值之一：0（像素点）；1（0.1毫米）；2（0.01毫米）；3（0.01英寸）；4（0.001英寸）；5（1/1440英寸）。





例如：

打印机\_属性.绘画单位 = 2

程序执行后会将“打印机\_属性”的绘画单位设为0.01毫米。

## 2) “画笔类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定在打印机绘画时的画笔类型。本属性可以是以下值之一：0（空笔）；

1（直线）；2（划线）；3（点线）；4（内划线）；5（双点划线）；6（内直线）。

例如：

打印机\_属性.画笔类型 = 3

程序执行后将“打印机\_属性”的画笔类型设为点线形式。

## 3) “画出方式”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打印机画笔的画出方式。

有关各种画出方式的说明为：0（黑色：始终画出黑色）；1（非或笔：或笔的相反）；2（与非笔：背景色以及画笔反相二者共有颜色的组合）；3（非复制笔：复制笔的相反）；4（与笔非：画笔以及已有颜色反相二者共有颜色的组合）；5（反转：已有颜色的反相）；6（异或笔：画笔的颜色以及已有颜色的组合，只取其一）；7（非与笔：与笔的相反）；8（与笔：画笔和已有二者共有颜色的组合）；9（非异或笔：异或笔的相反）；10（无操作：原有保持不变。该设置实际上关闭画笔）；11（或非笔：已有颜色与画笔颜色反相的组合）；12（复制笔（缺省值）：画笔颜色）；13（或笔非：画笔颜色与已有颜色的反相的组合）；14（或笔：画笔颜色与已有颜色的组合）；15（白色：始终画出白色）。

例如：

打印机\_属性.画出方式 = 15

程序执行后将“打印机\_属性”的画笔类型设为始终白色的画笔。

## 4) “画笔粗细”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置打印机中画笔的粗细。单位为“绘画单位”属性中的值，如果为0，表示画笔的粗细为一个像素点。当画笔类型为划线、点线、点划线、双点划线时，本属性无效，画笔宽度始终为一个像素点。

例如：

打印机1.开始打印 ( , , , , )

打印机1.画笔粗细 = 10

打印机1.画直线 (0, 0, 100, 100)

打印机1.结束打印 ()



程序执行后将“打印机1”的画笔粗细设为10个单位，在纸上打印从(0, 0)到(100, 100)的一条直线。

#### 5) “画笔颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置打印机内画笔的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

打印机\_属性.画笔颜色 = #蓝色

程序执行后将“打印机\_属性”的画笔颜色设为蓝色。

#### 6) “刷子类型”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置打印机中刷子的类型。

本属性可以是以下值之一：0（空刷子）；1（颜色刷子）；2（右斜线）；3（直交叉线）；4（斜交叉线）；5（左斜线）；6（横线）；7（竖线）；8（5%色点）；9（10%色点）；10（20%色点）；11（25%色点）；12（30%色点）；13（40%色点）；14（50%色点）；15（60%色点）；16（70%色点）；17（75%色点）；18（80%色点）；19（90%色点）；20（浅色上对角线）；21（浅色下对角线）；22（深色上对角线）；23（深色下对角线）；24（宽上对角线）；25（宽下对角线）；26（浅色竖线）；27（浅色横线）；28（窄竖线）；29（窄横线）；30（深色竖线）；31（深色横线）；32（上对角虚线）；33（下对角虚线）；34（横虚线）；35（竖虚线）；36（小纸屑）；37（大纸屑）；38（之字形）；39（波浪线）；40（对角砖形）；41（横向砖形）；42（纺织物）；43（方格）；44（草皮）；45（虚格线）；46（点式菱形）；47（瓦形）；48（棚架）；49（球体）；50（小网格）；51（大网格）；52（小棋盘）；53（大棋盘）；54（空心菱形）；55（实心菱形）。

例如：

打印机1.开始打印 ( , , , , )

打印机1.刷子类型 = 14

打印机1.刷子颜色 = #绿色

打印机1.填充矩形 (0, 0, 100, 100)

打印机1.结束打印 ()

程序执行后打印一个半透明绿色的矩形。

#### 7) “刷子颜色”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于设置打印机内刷子的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。



例如：

```
打印机1.开始打印 ( , , , , )
打印机1.刷子类型 = 14
打印机1.刷子颜色 = #绿色
打印机1.填充矩形 (0, 0, 100, 100)
打印机1.结束打印 ()
```

程序执行后打印一个半透明绿色的矩形。

### 8) “文本颜色” 属性

**属性类型：** 整数型； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于设置打印机内写出文本的颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

变量名	类型	静态	数组	备注
变量	字体			

```
变量.字体名称 = “黑体”
变量.字体大小 = 30
打印机1.开始打印 ( , , , , )
打印机1.文本颜色 = #黄色
打印机1.文本背景颜色 = #黑色
打印机1.字体 = 变量
打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后将“打印机1”的文本颜色设为黄色；将“打印机1”的文本背景色设为黑色；改变“打印机1”写出文字时的字体；在打印的当前位置写出一行文本，内容为“易语言”。

### 9) “文本背景颜色” 属性

**属性类型：** 整数型； **有效范围：** 设计时、编程时； **编程时权限：** 使用、读取、更改  
本属性用于设置打印机内写出文本的背景颜色。属性值为RGB颜色值，可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

变量名	类型	静态	数组	备注
变量	字体			

```
变量.字体名称 = “黑体”
变量.字体大小 = 30
打印机1.开始打印 ( , , , , )
打印机1.文本颜色 = #黄色
打印机1.文本背景颜色 = #黑色
打印机1.字体 = 变量
打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```





程序执行后将“打印机1”的文本颜色设为黄色；将“打印机1”的文本背景色设为黑色；改变“打印机1”写出文字时的字体；在打印的当前位置写出一行文本，内容为“易语言”。

#### 10) “字体”属性

**属性类型：**字体；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于设置打印机内写出文本的字体。

例如：

变量名	类型	静态	数组	备注
变量	字体			

```
变量.字体名称 = “黑体”  
变量.字体大小 = 30  
打印机1.开始打印 ( , , , , )  
打印机1.文本颜色 = #黄色  
打印机1.文本背景颜色 = #黑色  
打印机1.字体 = 变量  
打印机1.写文本行 (“易语言”)  
打印机1.结束打印 ()
```

程序执行后将“打印机1”的文本颜色设为黄色；将“打印机1”的文本背景色设为黑色；改变“打印机1”写出文字时的字体；在打印的当前位置写出一行文本，内容为“易语言”。

#### 11) “打印作业名”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

在打印过程中，本属性的内容将显示在由系统自动提供的打印状态对话框及 Windows 打印管理器中。如果为空，默认为“未命名打印作业”。需要注意的是，编程时在调用“开始打印”命令前更改本属性，才会在下次打印时奏效。

例如：

```
打印机1.打印作业名 = “新的任务”  
打印机1.开始打印 ( , , , , )  
打印机1.结束打印 ()
```

程序执行后设置下次打印作业名为“新的任务”；执行一次打印任务。

#### 12) “打印份数”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用作为打印设置对话框中的打印份数编辑框提供初始值，当用户从打印设置对话框中确认返回后，属性值提供用户所实际设置的需要打印份数。需了解的是：如果被选定打印机硬件支持多份拷贝功能，无论用户设置打印份数为多少，打印设置对话框永远返





回1，多份打印被转交给打印机硬件去自动完成。如果在进入打印设置对话框前置属性值为-1，表明现行易程序不支持多份打印，此时系统会自动检查当前所选择打印机硬件，如果该打印机硬件支持多份拷贝，将允许用户指定打印份数并启用硬件多份拷贝功能，否则不允许，打印设置对话框返回后属性值保持不变。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.打印份数 = 3
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后打印3份内容为“易语言”的文件。

### 13) “多份打印方式”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框中“份数”组中的“分页”选择框提供初始值，当用户从打印设置对话框中确认返回后，属性值提供用户所实际设置的值。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.打印份数 = 3
打印机1.多份打印方式 = 真
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后使用自动分页方式打印3份内容为“易语言”的文件。

### 14) “最小页号”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框提供最小页号值，如果在进入打印设置对话框前置属性值为-1，打印设置对话框中将不允许用户设置页打印范围。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.最小页号 = 1
打印机1.最大页号 = 5
打印机1.首页页号 = 2
打印机1.末页页号 = 3
打印机1.开始打印 ( , #A4纸, #纵向, , )
» 打印机1.写文本行 (“1”)
```





```

打印机1.换页 0
» 打印机1.写文本行 (“2”)
打印机1.换页 0
» 打印机1.写文本行 (“3”)
打印机1.换页 0
» 打印机1.写文本行 (“4”)
打印机1.换页 0
» 打印机1.写文本行 (“5”)
打印机1.结束打印 0

```

程序执行后使用自定义设置打印5页中的第2页和第3页。

#### 15) “最大页号” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框提供最大页号值。如果置为 -1，表示最大页号不受限制。如果“最小页号”属性值被置为 -1，本属性无意义。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```

打印机1.最小页号 = 1
打印机1.最大页号 = 5
打印机1.首页页号 = 2
打印机1.末页页号 = 3
打印机1.开始打印 ( , #A4纸, #纵向, , )
» 打印机1.写文本行 (“1”)
打印机1.换页 0
» 打印机1.写文本行 (“2”)
打印机1.换页 0
» 打印机1.写文本行 (“3”)
打印机1.换页 0
» 打印机1.写文本行 (“4”)
打印机1.换页 0
» 打印机1.写文本行 (“5”)
打印机1.结束打印 0

```

程序执行后使用自定义设置打印5页中的第2页和第3页。

#### 16) “首页页号” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框中的首打印页页号编辑框提供初始值，当用户从打印设置对话框中确认返回后，属性值提供用户所实际设置的首打印页号，此值将在“最小页号”属性值及“最大页号”属性值之间。但如果用户选择了全部打印或者打印选择区，本属性值将为 -1。如果“最小页号”属性值被置为 -1，本属性无意义。如果使用“开始打





印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.最小页号 = 1
打印机1.最大页号 = 5
打印机1.首页页号 = 2
打印机1.末页页号 = 3
打印机1.开始打印 ( , #A4纸, #纵向, , )
» 打印机1.写文本行 (“1”)
打印机1.换页 ()
» 打印机1.写文本行 (“2”)
打印机1.换页 ()
» 打印机1.写文本行 (“3”)
打印机1.换页 ()
» 打印机1.写文本行 (“4”)
打印机1.换页 ()
» 打印机1.写文本行 (“5”)
打印机1.结束打印 ()
```

程序执行后使用自定义设置打印5页中的第2页和第3页。

#### 17) “末页页号”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框中的末打印页页号编辑框提供初始值，如果置为 -1，表示不提供。当用户从打印设置对话框中确认返回后，属性值提供用户所实际设置的末打印页号，此值将在“首页页号”属性值及“最大页号”属性值之间。但如果用户选择了全部打印或者打印选择区，本属性值将为 -1。如果“最小页号”属性值被置为 -1，本属性无意义。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.最小页号 = 1
打印机1.最大页号 = 5
打印机1.首页页号 = 2
打印机1.末页页号 = 3
打印机1.开始打印 ( , #A4纸, #纵向, , )
» 打印机1.写文本行 (“1”)
打印机1.换页 ()
» 打印机1.写文本行 (“2”)
打印机1.换页 ()
» 打印机1.写文本行 (“3”)
打印机1.换页 ()
» 打印机1.写文本行 (“4”)
打印机1.换页 ()
```





```
打印机1.写文本行 (“5”)
打印机1.结束打印 ()
```

程序执行后使用自定义设置打印5页中的第2页和第3页。

#### 18) “打印选择区” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于为打印设置对话框中“选择的范围”选择框提供初始值，为 1 表示真，为 0 表示假。当用户从打印设置对话框中确认返回后，属性值提供用户所实际设置的值。如果在进入打印设置对话框前置属性值为 -1，打印设置对话框中将不允许用户选取该选择框。如果使用“开始打印”命令进入打印时没有调用打印设置对话框，本属性值保持不变。

例如：

```
打印机1.打印选择区 = 1
打印机1.开始打印 ( , , , , )
打印机1.结束打印 ()
```

设置打印设置对话框中选中“选择的范围”选项。

#### 19) “左边空” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打印时打印纸左边所留空白的尺寸，使用现行绘画单位，由系统自动提供支持。

例如：

```
打印机1.左边空 = 100
打印机1.顶边空 = 100
打印机1.右边空 = 100
打印机1.底边空 = 100
打印机1.开始打印 ( , , , , )
打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后设置“打印机1”打印时在各边自动预留100个当前单位空白；在纸上打印“易语言”三个字。

#### 20) “顶边空” 属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打印时打印纸顶边所留空白的尺寸，使用现行绘画单位，由系统自动提供支持。



例如：

```
打印机1.左边空 = 100
打印机1.顶边空 = 100
打印机1.右边空 = 100
打印机1.底边空 = 100
打印机1.开始打印 ( , , , , )
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后设置“打印机1”打印时在各边自动预留100个当前单位空白；在纸上打印“易语言”三个字。

#### 21) “右边空”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打印时打印纸右边所留空白的尺寸，使用现行绘画单位，由系统自动提供支持。

例如：

```
打印机1.左边空 = 100
打印机1.顶边空 = 100
打印机1.右边空 = 100
打印机1.底边空 = 100
打印机1.开始打印 ( , , , , )
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后设置“打印机1”打印时在各边自动预留100个当前单位空白；在纸上打印“易语言”三个字。

#### 22) “底边空”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定打印时打印纸底边所留空白的尺寸，使用现行绘画单位，由系统自动提供支持。

例如：

```
打印机1.左边空 = 100
打印机1.顶边空 = 100
打印机1.右边空 = 100
打印机1.底边空 = 100
打印机1.开始打印 ( , , , , )
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后设置“打印机1”打印时在各边自动预留100个当前单位空白；在纸上打印





“易语言”三个字。

### 23) “当前页号”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性提供了现行打印页的页号，由系统自行维护。页号值从“首页页号”属性值开始，如果“最小页号”属性值被置为-1，页号值从1开始。

在以下情况下属性值被自动增一：1 [使用“写文本行”命令打印一行文本并且要打印文本行的高度当前页容纳不下（此时将会首先自动换页）]；2（使用“换页”命令）。本属性仅在打印作业过程中有效。

例如：

```
打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.当前页号)
» 打印机1.写文本行 (“第一页”)
打印机1.换页 ()
» 输出调试文本 (打印机1.当前页号)
» 打印机1.写文本行 (“第二页”)
打印机1.结束打印 ()
```

程序执行后使用打印机打印两页内容，内容为“第一页”、“第二页”；换页前在输出面板中显示当前所操作的页号。

### 24) “已打印份数”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性提供现行所已经打印的份数（含当前份），由系统自行维护。在调用“开始下一份”命令后，本属性值被自动加一。

例如：

```
打印机1.打印份数 = 3
打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.已打印份数)
» 打印机1.写文本行 (“易语言”)
打印机1.开始下一份 ()
» 输出调试文本 (打印机1.已打印份数)
打印机1.结束打印 ()
```

程序执行后打印内容为“易语言”的文件3份；在输出面板中显示当前已打印的份数。

### 25) “打印区宽度”属性

**属性类型：**整数型；**有效范围：**编程时；**编程时权限：**使用、读取

本属性提供了打印纸实际可打印区域宽度减去左右边空后的值，使用现行绘画单位，仅在打印作业过程中有效。



例如：

```

打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.打印区宽度)
» 输出调试文本 (打印机1.打印区高度)
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
    
```

程序执行后使用打印机打印内容为“易语言”的文件；在输出面板中显示当前可用打印区域的宽度和高度。

## 26) “打印区高度” 属性

**属性类型：** 整数型； **有效范围：** 编程时； **编程时权限：** 使用、读取

本属性提供了打印纸实际可打印区域高度减去上下边空后的值，使用现行绘画单位，仅在打印作业过程中有效。

例如：

```

打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.打印区宽度)
» 输出调试文本 (打印机1.打印区高度)
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
    
```

程序执行后使用打印机打印内容为“易语言”的文件；在输出面板中显示当前可用打印区域的宽度和高度。

## 27) “打印被中止” 属性

**属性类型：** 逻辑型； **有效范围：** 编程时； **编程时权限：** 使用、读取

如果在打印时使用了系统所提供的打印状态对话框，并且在打印过程中用户已经选择中止打印取消了打印作业（系统自动支持），本属性值为真，否则为假。易程序在进行打印的过程中一旦发现本属性值为真，应该退出此过程。

例如：

```

打印机1.开始打印 ( , , , , )
... 如果真 (打印机1.打印被中止 = 假)
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
    
```

程序执行后判断打印是否已被终止。如果没有终止则在当前位置打印“易语言”。

## 4.11.4.2 打印机命令

### 1) “取设备句柄” 命令

命令原型为：

<整数型> 对象. 取设备句柄 ()



如当前用户程序正在进行打印作业，则返回对应的打印机设备句柄（即HDC），否则返回 0。本命令一般配合Windows GDI类API使用。

例如：

变量名	类 型	静态	数组	备 注
rt	RECT			
hbr	整数型			

```

打印机1.开始打印 ( , , , , )
rt.top = 100
rt.left = 100
rt.right = 900
rt.bottom = 900
hbr = CreateSolidBrush (#绿色)
FillRect (打印机1.取设备句柄 (), rt, hbr)
DeleteObject (hbr)
打印机1.结束打印 ()
    
```

程序执行后将在“打印机1”当前所操作纸中使用绿色填充（100，100）到（900，900）的矩形。

本例程源码见随书光盘中 “\图书例程\第四章\打印机\取设备句柄.e”

2) “开始打印” 命令

命令原型为：

<逻辑型> 对象. 开始打印 （ [逻辑型 是否调用打印设置对话框] ， [逻辑型 是否显示打印状态对话框] ， [通用型 纸张] ， [整数型 纸张方向] ， [整数型 纸张宽度] ， [整数型 纸张高度] ）

本命令用于通知打印机开始进入打印状态。返回“真”表示已经成功地进入了打印作业，进入打印作业后文本写出位置将被重置为 0 。如果执行本命令时已经在打印作业中或者用户在打印设置对话框中选择了取消，返回“假”。进入打印后必须调用“结束打印”或者“取消打印”命令来退出打印。进入打印前易程序进行的所有打印绘画操作都将被忽略。

参数<1>指定了在进入打印之前是否先调用打印设置对话框对打印机及相关打印参数进行设置。如省略本参数，默认为“真”。如果参数值为“假”，将自动使用Windows 系统默认打印机，但如果取 Windows 系统默认打印机失败，仍然会调用打印设置对话框。

参数<2>指定了在打印过程中是否显示由系统自动提供的打印状态对话框。用户如果在该打印状态对话框中选择了取消打印，系统将自动中止并取消该打印作业，并置“打印被中止”属性值为真。如省略本参数，默认为“真”。

参数<3>指定可所使用的打印纸类型。



可以为以下常量值之一：

-1（自定义纸张：必须在后面参数中指定自定义纸张的大小）；0（#默认纸）；1（#A3纸：297×420毫米）；2（#A4纸：210×297毫米）；3（#A5纸：148×210毫米）；4（#B4纸：250×354毫米）；5（#B5纸：182×257毫米）；6（#四开：215×275毫米）；7（#十六开：146×215毫米）；8（#三十二开：97×151毫米）；9（#信纸：216×279毫米）；10（#法律用纸：216×355毫米）；11（#行政用纸：184×266毫米）；12（#声明：140×216毫米）；13（#小报：279×432毫米）；14（#笔记：216×279毫米）；15（#账本：432×279毫米）；16（#对开纸：216×330毫米）。

除了以上基本类型纸张，还支持以下扩展类型，这些类型值和其所对应纸张宽度与高度（单位毫米）如下：

1001: 216×279、1002: 279×432、1003: 432×279、1004: 216×355、1005: 140×216、1006: 184×267、1007: 297×420、1008: 210×297、1010: 148×210、1011: 257×364、1012: 182×257、1013: 216×330、1014: 215×275、1015: 254×355、1016: 279×432、1017: 216×279、1018: 98×225、1019: 105×241、1020: 114×263、1021: 121×279、1022: 127×292、1023: 432×558、1024: 558×863、1025: 863×1117、1026: 110×220、1027: 162×229、1028: 324×458、1029: 229×324、1030: 114×162、1031: 114×229、1032: 250×353、1033: 176×250、1034: 176×125、1035: 110×230、1036: 98×190、1037: 92×165、1038: 378×279、1039: 216×305、1040: 216×330、1041: 250×353、1042: 100×148、1043: 228×279、1044: 254×279、1045: 381×279、1046: 220×220、1049: 241×305、1050: 241×381、1051: 305×457、1052: 235×322、1053: 216×279、1054: 210×297、1055: 241×305、1056: 227×356、1057: 305×487、1058: 216×322、1059: 210×330、1060: 148×210、1061: 182×257、1062: 322×445、1063: 174×235、1064: 201×276、1065: 420×594、1066: 297×420、1067: 322×445、1068: 200×148、1069: 105×148、1070: 240×332、1071: 216×277、1072: 120×235、1073: 90×205、1074: 279×216、1075: 420×297、1076: 297×210、1077: 210×148、1078: 364×257、1079: 257×182、1080: 148×100、1081: 148×200、1082: 148×105、1083: 332×240、1084: 277×216、1085: 235×120、1086: 205×90、1087: 128×182、1088: 182×128、1089: 305×279、1090: 105×235、1091: 235×105、1092: 188×260、1093: 130×184、1094: 140×203、1095: 102×165、1096: 102×176、1097: 125×176、1098: 110×208、1099: 110×220、1100: 120×230、1101: 160×230、1102: 120×309、1103: 229×324、1104: 324×458、1105: 260×188、1106: 184×130、1107: 203×140、1108: 165×102、1109: 176×102、1110: 176×125、1111: 208×110、1112: 220×110、1113: 230×120、1114: 230×160、1115: 309×120、1116: 324×229、1117: 458×324





**注解：**如果所选择纸张得不到打印机的支持，打印机将会自动选择最接近的纸张。

此外，在Windows NT 4.0及以上系统下，还可以为本参数指定自定义纸张名称文本（必需已经存在，可通过“置自定义纸张类型”命令进行动态添加或修改）；在Windows 95/98/Me系统下，如果为本参数提供了自定义纸张名称，则本参数被忽略。如省略本参数，默认为“#默认纸”。

参数<4>指定了打印纸的放置方向。可以为以下常量值之一：0（#纵向）；1（#横向）。如省略本参数，默认为“#纵向”。

参数<5>指定了自定义纸张的宽度，单位为0.1毫米。只有当前面的“纸张”参数给定值为-1时，本参数才有作用且必须提供具体值。

参数<6>指定了自定义纸张的高度，单位为0.1毫米。只有当前面的“纸张”参数给定值为-1时，本参数才有作用且必须提供具体值。

例如：

```
打印机1.开始打印 (真, 真, #A4纸, #纵向, , )
打印机1.写文本行 ("易语言")
打印机1.结束打印 ()
```

程序执行后使用打印机打印“易语言”三个字。

### 3) “结束打印”命令

命令原型为：

<无返回值> 对象. 结束打印 ()

如果当前已经进入了打印作业，调用本命令可完成此作业。结束打印后易程序进行的后续打印绘画操作都将被忽略。

例如：

```
打印机1.开始打印 (真, 真, #A4纸, #纵向, , )
打印机1.写文本行 ("易语言")
打印机1.结束打印 ()
```

程序执行后使用打印机打印“易语言”三个字。

### 4) “取消打印”命令

命令原型为：

<无返回值> 对象. 取消打印 ()

如果当前已经进入了打印作业，调用本命令可立即中止且取消该作业。如果操作系统的打印管理器正在处理该打印作业（打印管理器正在运行并且允许后台打印），那么该作业将被删除且使打印机不接收任何信息。如果打印管理器不是正在处理该作业（没有选用后台打印），部分或全部数据可能在此之前已发送到打印机。此时，打印机驱动程序将尽可能使打印机立即复位并终止该作业。取消打印后易程序进行的后续打印绘画操作都将被忽略。





例如：

```
打印机1.开始打印 (真, 真, #A4纸, #纵向, , )
打印机1.填充矩形 (0, 0, 500, 500)
打印机1.结束打印 ()
打印机1.取消打印 ()
```

程序执行后使用打印机填充一个矩形；立即取消打印机正在打印的任务（未完成的打印工作将被抛弃）。

#### 5) “换页”命令

命令原型为：

<逻辑型> 对象. 换页 ()

调用本命令使打印机完成当前页的打印，文本写出位置被重置为 0，并走纸到下一页左上角。如果换页成功返回“真”，否则将自动取消打印并返回“假”。

例如：

```
打印机1.开始打印 (真, 真, #A4纸, #纵向, , )
打印机1.写文本行 (“第一页”)
打印机1.换页 ()
打印机1.写文本行 (“第二页”)
打印机1.结束打印 ()
```

程序执行后会分别在两页上打印“第一页”、“第二页”。

#### 6) “开始下一份”命令

命令原型为：

<逻辑型> 对象. 开始下一份 ()

调用本命令后将结束在当前页上的打印作业，并将当前打印份数加一，当前页号置回到首页页号，以进入下一份拷贝的打印。如果成功返回“真”，失败将自动取消打印并返回“假”。

例如：

```
打印机1.打印份数 = 3
打印机1.开始打印 ( , , , , )
输出调试文本 (打印机1.已打印份数)
打印机1.写文本行 (“易语言”)
打印机1.开始下一份 ()
输出调试文本 (打印机1.已打印份数)
打印机1.结束打印 ()
```

程序执行后打印内容为“易语言”的文件3份；在输出面板中显示当前已打印的份数。

#### 7) “画点”命令

命令原型为：





<无返回值> 对象. 画点 (整数型 点横坐标, 整数型 点纵坐标, 整数型 欲画入点的颜色值)

本命令用于在打印机指定位置使用指定颜色填充一个点。

参数<1>指定了欲填充点的横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了欲填充点的纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了用于填充点的颜色值。颜色值可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量，如“#红色”等。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.画点 (10, 10, #红色)
打印机1.结束打印 ()
```

程序执行后在“打印机1”的(10, 10)坐标处使用红色填充一个点。

#### 8) “画直线”命令

命令原型为：

<无返回值> 对象. 画直线 (整数型 起始点横坐标, 整数型 起始点纵坐标, 整数型 结束点横坐标, 整数型 结束点纵坐标)

本命令用于使用当前画笔在打印机上画出一条直线。

参数<1>指定了直线起始点横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了直线起始点纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了直线结束点横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了直线结束点纵坐标。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.画直线 (0, 0, 100, 100)
打印机1.结束打印 ()
```

程序执行后在“打印机1”中从(0, 0)到(100, 100)画一条直线。

#### 9) “画椭圆”命令

命令原型为：

<无返回值> 对象. 画椭圆 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标)

本命令用于使用当前刷子在打印机上填充一个椭圆。本命令的参数指定了外切于椭圆的矩形区域。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于打印纸左上角。





参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )  
打印机1.画椭圆 (0, 0, 100, 50)  
打印机1.结束打印 ()
```

程序执行后在“打印机1”中使用当前刷子填充内切于(0, 0)，(100, 50)矩形的椭圆。

#### 10) “画弧线”命令

命令原型为：

<无返回值> 对象.画弧线 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前画笔在打印机上画出一条弧线。本命令所画图形相当于一个内切矩形的椭圆形边线的一部分。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于打印纸左上角。

参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )  
打印机1.画弧线 (0, 0, 100, 100, 0, 0, 50, 50)  
打印机1.结束打印 ()
```

执行后打印效果如图4-36所示。



图4-36 画弧线效果





## 11) “画弦”命令

命令原型为:

<无返回值> 对象. 画弦 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前刷子在打印机上填充一个椭圆的弦。椭圆的轮廓由外切矩形区域指定。

参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位, 相对于打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位, 相对于打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位, 相对于打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位, 相对于打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位, 相对于打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位, 相对于打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位, 相对于打印纸左上角。

参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位, 相对于打印纸左上角。

例如:

```
打印机1.开始打印 ( , , , , )  
打印机1.画弦 (0, 0, 100, 100, 0, 0, 50, 50)  
打印机1.结束打印 ()
```

执行后打印效果如图4-37所示。



图4-37 画弦效果

## 12) “画饼”命令

命令原型为:

<无返回值> 对象. 画饼 (整数型 椭圆左上角横坐标, 整数型 椭圆左上角纵坐标, 整数型 椭圆右下角横坐标, 整数型 椭圆右下角纵坐标, 整数型 弧线起始点横坐标, 整数型 弧线起始点纵坐标, 整数型 弧线终止点横坐标, 整数型 弧线终止点纵坐标)

本命令用于使用当前刷子在打印机上填充一个椭圆, 并切掉原切角区域(V型)。椭圆的轮廓由外切矩形区域指定。





参数<1>指定了外切矩形左上角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了外切矩形左上角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了外切矩形右下角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了外切矩形右下角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<5>指定了弧线起始点横坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于打印纸左上角。

参数<6>指定了弧线起始点纵坐标。弧线从椭圆中心点至此起始点的直线与椭圆圆弧相交处开始绘制。使用当前绘画单位，相对于打印纸左上角。

参数<7>指定了弧线终止点横坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于打印纸左上角。

参数<8>指定了弧线终止点纵坐标。弧线在椭圆中心点至此终止点的直线与椭圆圆弧相交处结束绘制。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.画饼 (0, 0, 100, 100, 0, 0, 100, 50)
打印机1.结束打印 ()
```

执行结果如图4-38所示。



图4-38 画饼效果

### 13) “画矩形”命令

命令原型为：

<无返回值> 对象.画矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标)

本命令用于使用当前画笔在打印机上画出一个矩形，矩形的内部使用当前刷子填充。

参数<1>指定了矩形左上角的横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了矩形左上角的纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了矩形右下角的横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了矩形右下角的纵坐标。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.画矩形 (0, 0, 100, 100)
打印机1.结束打印 ()
```

程序执行后在“打印机1”中(0, 0)位置画一个100×100的矩形，并填充矩形区域。





## 14) “画渐变矩形”命令

命令原型为:

<无返回值> 对象. 画渐变矩形 (整数型 矩形区域左边, 整数型 矩形区域顶边, 整数型 矩形区域宽度, 整数型 矩形区域高度, [整数型 渐变方向], 整数型 首渐变颜色, 整数型 其他渐变颜色, ...)

本命令用于在打印机上使用指定方式填充一个渐变矩形区域。本命令的最后一个参数可以重复扩展以指定更多渐变关键色。

参数<1>指定了欲填充矩形的起始横坐标。

参数<2>指定了欲填充矩形的起始纵坐标。

参数<3>指定了欲填充矩形的宽度。

参数<4>指定了欲填充矩形的高度。

参数<5>指定了形成渐变的样式。

参数值可以为以下常量值之一：1（#从上到下）；2（#从左到右）；3（#从左上到右下）；4（#从右上到左下）；5（#从下到上）；6（#从右到左）；7（#从右下到左上）；8（#从左下到右上）。如果本参数被省略，默认为“#从左到右”。

参数<6>指定了第一个关键色的颜色值。

参数<7>及以后的参数指定了渐变时的关键颜色值。

例如:

```
打印机1. 开始打印 ( , , , , )
打印机1. 画渐变矩形 (0, 0, 100, 100, #从左上到右下, #红色, #绿色, #蓝色)
打印机1. 结束打印 ()
```

执行后打印效果如图4-39所示。



图4-39 画渐变矩形效果

## 15) “填充矩形”命令

命令原型为:

<无返回值> 对象. 填充矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标)

本命令用于将打印机上指定的矩形区域用当前刷子填充。本命令与“画矩形”命令区别在于有无边框。

参数<1>指定了欲填充矩形的左上角横坐标。使用当前绘画单位，相对于打印纸左上角。





参数<2>指定了欲填充矩形的左上角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了欲填充矩形的右下角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了欲填充矩形的右下角纵坐标。使用当前绘画单位，相对于打印纸左上角。

例如：

```
打印机1.开始打印 ( , , , , )  
打印机1.填充矩形 (0, 0, 100, 100)  
打印机1.结束打印 ()
```

程序执行后使用当前刷子填充“打印机1”中的(0, 0)到(100, 100)的区域。

#### 16) “画圆角矩形”命令

命令原型为：

<无返回值> 对象. 画圆角矩形 (整数型 矩形左上角横坐标, 整数型 矩形左上角纵坐标, 整数型 矩形右下角横坐标, 整数型 矩形右下角纵坐标, 整数型 圆角宽度, [整数型 圆角高度])

本命令用于使用当前画笔在打印机上画出一个圆角矩形，圆角矩形的内部使用当前刷子填充。

参数<1>指定了矩形的左上角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了矩形的左上角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了矩形的右下角横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了矩形的右下角纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<5>指定了圆角所在扇形的宽度。使用当前绘画单位。

参数<6>指定了圆角所在扇形的高度。使用当前绘画单位。如果省略本参数，默认与圆角宽度相等。

例如：

```
打印机1.开始打印 ( , , , , )  
打印机1.画圆角矩形 (0, 0, 100, 100, 20, 20)  
打印机1.结束打印 ()
```

执行后打印效果如图4-40所示。



图4-40 画圆角矩形效果





## 17) “画多边形”命令

命令原型为:

&lt;无返回值&gt; 对象. 画多边形 (整数型数组 多边形顶点, [整数型 顶点数目])

本命令用于在打印机内使用当前画笔画一个多边形, 并用当前刷子填充这个区域。如果所画的多边形没有闭合, 将自动闭合。

参数<1>指定了多边形顶点坐标组。提供参数数据时只能提供数组数据, 提供的数组在内部将自动转换为一维数组。数组结构为: 顶点1横坐标, 顶点1纵坐标, 顶点2横坐标, 顶点2纵坐标……以此类推。使用当前绘画单位, 相对于打印纸左上角。

参数<2>指定了多边形顶点的总数目。如果被省略, 默认为数组中所记录的顶点数目, 既根据第二个参数成员数自动计算。

例如:

变量名	类型	静态	数组	备注
变量	整数型		3,2	

变量 [1] [1] = 50

变量 [1] [2] = 0

变量 [2] [1] = 0

变量 [2] [2] = 50

变量 [3] [1] = 100

变量 [3] [2] = 50

打印机1. 开始打印 ( , , , , )

打印机1. 画多边形 (变量, 3)

打印机1. 结束打印 ()

执行后打印效果如图4-41所示。

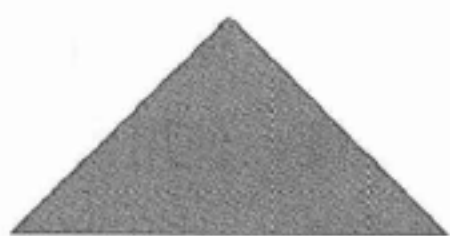


图4-41 画多边形效果

## 18) “置写出位置”命令

命令原型为:

&lt;无返回值&gt; 对象. 置写出位置 ([整数型 横向写出位置], [整数型 纵向写出位置])

本命令用于设置下次使用“写文本行”或“写出”命令输出数据时的位置。

参数<1>指定了新写出位置的横坐标。如果本参数被省略, 则使用现行横向写出位置。使用当前绘画单位, 相对于打印纸左上角。

参数<2>指定了新写出位置的纵坐标。如果本参数被省略, 则使用现行纵向写出位置。使用当前绘画单位, 相对于打印纸左上角。





例如：

```
打印机1.开始打印 ( , , , , )
打印机1.置写出位置 (100, 100)
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后，设置新的文字写出位置为（100,100）；在“打印机1”中使用当前字体及字体颜色写出“易语言”。

#### 19) “写文本行”命令

命令原型为：

<无返回值> 对象. 写文本行 ([通用型 欲写出数据], ...)

本命令用于在当前写出位置写出指定的文本、数值、逻辑值或日期时间，并将现行写出位置调整到下行行首。

参数<1>及后续参数指定了写出的内容。参数值只能为文本、数值、逻辑值或日期时间。如果本参数被省略，则写出一个空行。

例如：

```
打印机1.开始打印 ( , , , , )
打印机1.置写出位置 (100, 100)
» 打印机1.写文本行 (“易语言”)
打印机1.结束打印 ()
```

程序执行后，设置新的文字写出位置为（100, 100）；在“打印机1”中使用当前字体及字体颜色写出“易语言”。

#### 20) “写出”命令

命令原型为：

<无返回值> 对象. 写出 ([通用型 欲写出数据], ...)

本命令用于在当前写出位置处写出指定的文本、数值、逻辑值或日期时间。自动调整现行写出位置到所写出数据的末位置。

参数<1>及后续参数指定了欲写出的内容。参数值只能为文本、数值、逻辑值或日期时间。如果本参数被省略，则不写出任何内容。

例如：

```
打印机1.开始打印 ( , , , , )
» 打印机1.写出 (“易”, “语”, “言”)
打印机1.结束打印 ()
```

程序执行后在“打印机1”当前写出位置分三次写出“易”、“语”、“言”。

#### 21) “定位写出”命令

命令原型为：



<无返回值> 对象. 定位写出 ([整数型 横向写出位置], [整数型 纵向写出位置], 通用型 欲写出数据, ...)

本命令用于在指定写出位置处写出指定的文本、数值、逻辑值或日期时间，不改变现行写出位置。

参数<1>指定了写出文字的起始横坐标。如果本参数被省略，则使用现行横向写出位置。使用当前绘画单位，相对于打印纸左上角。

参数<2>指定了写出文字的起始纵坐标。如果本参数被省略，则使用现行纵向写出位置。使用当前绘画单位，相对于打印纸左上角。

参数<3>及后续参数指定了欲写出的内容。参数值只能为文本、数值、逻辑值或日期时间。

例如：

```

打印机1.开始打印 ( , , , , )
» 打印机1.定位写出 (50, 50, “易语言”)
打印机1.结束打印 ()
    
```

程序执行后在“打印机1”的(50, 50)位置写出“易语言”。

### 22) “取宽度”命令

命令原型为：

<整数型> 对象. 取宽度 (通用型 欲取其宽度或高度的数据)

本命令用于获取指定数据在打印机写出时的宽度，使用当前绘画单位。如果调用本命令时尚未进入打印，将返回0。

参数<1>指定了欲测试的数据。参数值只能为文本、数值、逻辑值或日期时间。

例如：

变量名	类型	静态	数组	备注
打印内容	文本型			

```

打印内容 = “易语言”
打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.取宽度 (打印内容))
» 输出调试文本 (打印机1.取高度 (打印内容))
打印机1.结束打印 ()
    
```

程序执行后在输出面板显示“易语言”在打印机写出时所需要的宽度和高度。

### 23) “取高度”命令

命令原型为：

<整数型> 对象. 取高度 (通用型 欲取其宽度或高度的数据)

本命令用于获取指定数据在打印机写出时的高度，使用当前绘画单位。如果调用本命令时尚未进入打印，将返回0。



参数<1>指定了欲测试的数据。参数值只能为文本、数值、逻辑值或日期时间。

例如：

变量名	类型	静态	数组	备注
打印内容	文本型			

打印内容 = “易语言”

打印机1.开始打印 ( , , , , )

» 输出调试文本 (打印机1.取宽度 (打印内容))

» 输出调试文本 (打印机1.取高度 (打印内容))

打印机1.结束打印 ()

程序执行后在输出面板显示“易语言”在打印机写出时所需要的高度和高度。

#### 24) “画图片”命令

命令原型为：

<无返回值> 对象.画图片 (整数型 图片号, 整数型 图片左边画出位置, 整数型 图片顶边画出位置, [整数型 图片画出宽度], [整数型 图片画出高度], [整数型 图片画出方法])

本命令用于将使用“载入图片”命令所载入的资源以指定尺寸、方式画到打印机指定位置。

参数<1>指定了使用“载入图片”命令所返回的图片号。

参数<2>指定了图片画出时位置的起始横坐标。使用当前绘画单位，相对于打印纸左上角。

参数<3>指定了图片画出时位置的起始纵坐标。使用当前绘画单位，相对于打印纸左上角。

参数<4>指定了画出图片时将图片缩放到的目标宽度。如果本参数被省略，则使用图片本身的宽度。使用当前绘画单位。

参数<5>指定了画出图片时将图片缩放到的目标高度。如果本参数被省略，则使用图片本身的高度。使用当前绘画单位。

参数<6>定义在将图像绘制到对象（画板/打印机）上时对图像执行的位操作。对“图标”类型图片无效。

可以为以下常量值之一或者其他自定义操作码：1（#拷贝）；2（#翻转拷贝）；3（#位异或）；4（#位或）；5（#位与）。

本参数也可以指定透明色，但必须是负颜色数值。如：

打印机1.画图片 (图片号, 0, 0, , , -取颜色值 (255, 255, 255))

此语句的作用就是在打印机上画出图片中除白色外的所有颜色。

如果省略本参数，默认为“#拷贝”。





例如：

变量名	类 型	静态	数组	备 注
图片号	整数型			

```

图片号 = 载入图片 ("C:\WINDOWS\Prairie Wind.bmp")
打印机1.开始打印 ( , , , , )
打印机1.画图片 (图片号, 0, 0, , , #拷贝)
打印机1.置写出位置 (0, 300)
» 打印机1.写文本行 (打印机1.取图片宽度 (图片号))
» 打印机1.写文本行 (打印机1.取图片高度 (图片号))
打印机1.结束打印 ()
    
```

程序执行后将“C:\WINDOWS\Prairie Wind.bmp”画在打印机（0，0）处；在图片下方显示图片的宽度与高度。

## 25) “取图片宽度” 命令

命令原型为：

<整数型> 对象. 取图片宽度 (整数型 图片号)

本命令用于获取指定图片的宽度。如果调用本命令时尚未进入打印，使用像素点为单位，否则使用当前绘画单位。

参数<1>指定了欲获取图片宽度对应的图片号。图片号由“载入图片”命令返回。

例如：

变量名	类 型	静态	数组	备 注
图片号	整数型			

```

图片号 = 载入图片 ("C:\WINDOWS\Prairie Wind.bmp")
打印机1.开始打印 ( , , , , )
打印机1.画图片 (图片号, 0, 0, , , #拷贝)
打印机1.置写出位置 (0, 300)
» 打印机1.写文本行 (打印机1.取图片宽度 (图片号))
» 打印机1.写文本行 (打印机1.取图片高度 (图片号))
打印机1.结束打印 ()
    
```

程序执行后将“C:\WINDOWS\Prairie Wind.bmp”画在打印机（0，0）处；在图片下方显示图片的宽度与高度。

## 26) “取图片高度” 命令

命令原型为：

<整数型> 对象. 取图片高度 (整数型 图片号)

本命令用于获取指定图片的高度。如果调用本命令时尚未进入打印，使用像素点为单位，否则使用当前绘画单位。

参数<1>指定了欲获取图片宽度对应的图片号。图片号由“载入图片”命令返回。



例如：

变量名	类 型	静态	数组	备 注
图片号	整数型			

```
图片号 = 载入图片 ("C:\WINDOWS\Prairie Wind.bmp")
打印机1.开始打印 ( , , , , )
打印机1.画图片 (图片号, 0, 0, , , #拷贝)
打印机1.置写出位置 (0, 300)
» 打印机1.写文本行 (打印机1.取图片宽度 (图片号))
» 打印机1.写文本行 (打印机1.取图片高度 (图片号))
打印机1.结束打印 ()
```

程序执行后将“C:\WINDOWS\Prairie Wind.bmp”画在打印机（0，0）处；在图片下方显示图片的宽度与高度。

27) “置自定义纸张类型” 命令

命令原型为：

<逻辑型> 对象. 置自定义纸张类型 (文本型 自定义纸张类型名称, 整数型 纸张宽度, 整数型 纸张高度, [整数型 上边距], [整数型 下边距], [整数型 左边距], [整数型 右边距])

添加或修改自定义纸张类型。本命令仅在Windows NT 4.0及以上系统下有效，且要求具有打印机驱动的完全控制权限。

**注解：** 太小或太大的纸张大小和页边距，可能不被特定的打印机所支持。本命令所设置的自定义纸张类型，可用于“开始打印（）”的第三个参数“纸张类型”。添加或修改后的自定义纸张类型可以在“当前默认打印机驱动所属计算机”的“开始-设置-打印机和传真-文件（菜单）-服务器属性（菜单）”中看到（仅限Windows NT 4.0及以上系统）。

参数<1>指定了自定义纸张类型名称。

参数<2>指定了自定义纸张的宽度。单位为 0.1 毫米。

参数<3>指定了自定义纸张的高度。单位为 0.1 毫米。

参数<4>指定了自定义纸张的打印上边距。单位为 0.1 毫米。如果本参数被省略，默认为0。

参数<5>指定了自定义纸张的打印下边距。单位为 0.1 毫米。如果本参数被省略，默认为0。

参数<6>指定了自定义纸张的打印左边距。单位为 0.1 毫米。如果本参数被省略，默认为0。

参数<7>指定了自定义纸张的打印右边距。单位为 0.1 毫米。如果本参数被省略，默认为0。





例如：

变量名	类型	静态	数组	备注
纸张宽度	整数型			
纸张高度	整数型			
上边距	整数型			
下边距	整数型			
左边距	整数型			
右边距	整数型			

打印机1. 置自定义纸张类型 (“易语言自定义纸张”, 100, 500, 10, 10, 10, 10)

打印机1. 取自定义纸张大小 (“易语言自定义纸张”, 纸张宽度, 纸张高度, 上边距, 下边距, 左边距, 右边距)

» 输出调试文本 (“纸张宽度:”)

» 输出调试文本 (纸张宽度)

» 输出调试文本 (#换行符)

» 输出调试文本 (“纸张高度:”)

» 输出调试文本 (纸张高度)

» 输出调试文本 (#换行符)

» 输出调试文本 (“上边距:”)

» 输出调试文本 (上边距)

» 输出调试文本 (#换行符)

» 输出调试文本 (“下边距:”)

» 输出调试文本 (下边距)

» 输出调试文本 (#换行符)

» 输出调试文本 (“左边距:”)

» 输出调试文本 (下边距)

» 输出调试文本 (#换行符)

» 输出调试文本 (“右边距:”)

» 输出调试文本 (右边距)

打印机1. 删除自定义纸张类型 (“易语言自定义纸张”)

程序执行后在系统内新建名称为“易语言自定义纸张”的新纸张类型；获取新建的纸张信息并显示在输出面板中；最后删除新建的纸张类型。

## 28) “删除自定义纸张类型”命令

命令原型为：

<逻辑型> 对象. 删除自定义纸张类型 (文本型 自定义纸张类型名称)

本命令用于删除指定自定义纸张类型。本命令仅在Windows NT 4.0及以上系统下有效，且要求具有打印机驱动的完全控制权限。

参数<1>指定了欲删除的自定义纸张类型名称。





例如：

变量名	类型	静态	数组	备注
纸张宽度	整数型			
纸张高度	整数型			
上边距	整数型			
下边距	整数型			
左边距	整数型			
右边距	整数型			

打印机1.置自定义纸张类型（“易语言自定义纸张”，100，500，10，10，10，10）

打印机1.取自定义纸张大小（“易语言自定义纸张”，纸张宽度，纸张高度，上边距，下边距，左边距，右边距）

» 输出调试文本（“纸张宽度：”）

» 输出调试文本（纸张宽度）

» 输出调试文本（#换行符）

» 输出调试文本（“纸张高度：”）

» 输出调试文本（纸张高度）

» 输出调试文本（#换行符）

» 输出调试文本（“上边距：”）

» 输出调试文本（上边距）

» 输出调试文本（#换行符）

» 输出调试文本（“下边距：”）

» 输出调试文本（下边距）

» 输出调试文本（#换行符）

» 输出调试文本（“左边距：”）

» 输出调试文本（左边距）

» 输出调试文本（#换行符）

» 输出调试文本（“右边距：”）

» 输出调试文本（右边距）

打印机1.删除自定义纸张类型（“易语言自定义纸张”）

程序执行后在系统内新建名称为“易语言自定义纸张”的新纸张类型；获取新建的纸张信息并显示在输出面板中；最后删除新建的纸张类型。

29) “取自定义纸张大小”命令

命令原型为：

<逻辑型> 对象.取自定义纸张大小（文本型 自定义纸张类型名称，[整数型变量 纸张宽度]，[整数型变量 纸张高度]，[整数型变量 上边距]，[整数型变量 下边距]，[整数型变量 左边距]，[整数型变量 右边距]）

本命令用于取指定自定义纸张类型的纸张大小及边距，并写入后面的六个参数中。如果指定的自定义纸张类型不存在，返回“假”。本命令仅在Windows NT 4.0及以上系统下



有效，且要求具有打印机驱动的完全控制权限。

参数<1>指定了欲获取信息的自定义纸张类型名称。

参数<2>用于提供变量以获取纸张的宽度数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

参数<3>用于提供变量以获取纸张的高度数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

参数<4>用于提供变量以获取纸张的上边距数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

参数<5>用于提供变量以获取纸张的下边距数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

参数<6>用于提供变量以获取纸张的左边距数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

参数<7>用于提供变量以获取纸张的右边距数据。可以被省略，提供参数数据时只能提供变量。单位为 0.1 毫米。

例如：

变量名	类 型	静态	数组	备 注
纸张宽度	整数型			
纸张高度	整数型			
上边距	整数型			
下边距	整数型			
左边距	整数型			
右边距	整数型			

打印机1. 置自定义纸张类型（“易语言自定义纸张”，100，500，10，10，10，10）

打印机1. 取自定义纸张大小（“易语言自定义纸张”，纸张宽度，纸张高度，上边距，下边距，左边距，右边距）

- » 输出调试文本（“纸张宽度：”）
- » 输出调试文本（纸张宽度）
- » 输出调试文本（#换行符）
- » 输出调试文本（“纸张高度：”）
- » 输出调试文本（纸张高度）
- » 输出调试文本（#换行符）
- » 输出调试文本（“上边距：”）
- » 输出调试文本（上边距）
- » 输出调试文本（#换行符）
- » 输出调试文本（“下边距：”）
- » 输出调试文本（下边距）
- » 输出调试文本（#换行符）
- » 输出调试文本（“左边距：”）



» 输出调试文本 (下边距)  
» 输出调试文本 (#换行符)  
» 输出调试文本 (“右边距:”)  
» 输出调试文本 (右边距)  
打印机1. 删除自定义纸张类型 (“易语言自定义纸张”)

程序执行后在系统内新建名称为“易语言自定义纸张”的新纸张类型；获取新建的纸张信息并显示在输出面板中；最后删除新建的纸张类型。

30) “取所有纸张类型” 命令

命令原型为：

<文本型数组> 对象. 取所有纸张类型 ()

取当前默认打印机所支持的所有纸张类型（包括系统定义纸张类型和自定义纸张类型），返回一维文本数组，其中存放了的各纸张类型的名称。本命令仅在Windows NT 4.0及以上系统下有效。

例如：

变量名	类 型	静态	数组	备 注
纸张类型组	文本型		0	
计次变量	整数型			

纸张类型组 = 打印机1. 取所有纸张类型 ()  
--- 计次循环首 (取数组成员数 (纸张类型组), 计次变量)  
» 输出调试文本 (纸张类型组 [计次变量])  
--- 计次循环尾 ()

程序执行后在输出面板中显示当前系统中所有纸张的类型名称。

31) “单位转换” 命令

命令原型为：

<整数型> 对象. 单位转换 (整数型 欲转换的坐标值, 整数型 欲转换坐标值的类型)

本命令用于将像素单位坐标值转换到当前绘画单位，或将当前绘画单位坐标值转换到像素单位。

**注解：** 在执行本命令之前，必须已经开始打印。

参数<1>提供欲转换的横向或纵向坐标值。

参数<2>指定了上一个参数的类型。参数可以为以下常量值之一：1（#横向绘画单位）；2（#纵向绘画单位）；3（#横向像素单位）；4（#纵向像素单位）。根据所提供的类型值，将进行相反的单位转换，即将绘画单位转换到像素单位，将像素单位转换到绘画单位。





例如：

```
打印机1.绘画单位 = 1
打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.单位转换 (50, #横向像素单位))
打印机1.结束打印 ()
```

程序执行后将“打印机1”的绘画单位设为0.1毫米；将50个横向绘画单位转换为像素值显示在输出面板中。

### 32) “取设备名称”命令

命令原型为：

<文本型> 对象. 取设备名称 ()

如果已经开始打印，则返回当前打印机设备的名称，否则返回空文本。

例如：

```
打印机1.开始打印 ( , , , , )
» 输出调试文本 (打印机1.取设备名称 ())
打印机1.结束打印 ()
```

程序执行后在输出面板中显示当前所使用的打印机的名称。

## 4.11.5 影像框

### 影像框属性

#### 1) “文件名”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定欲播放影像文件的名称。

例如：

```
影像框1.文件名 = "C:\WINDOWS\clock.avi"
影像框1.居中播放 = 假
影像框1.透明背景 = 真
影像框1.播放 = 真
影像框1.播放次数 = -1
```

程序执行后将“影像框1”当前播放的文件设为“C:\WINDOWS\clock.avi”；播放时设置影像位于影像框的左上；播放时透明影像框的背景；设置“影像框1”播放当前的影像；设置播放次数为循环播放。

#### 2) “居中播放”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定在播放影像时影像是否居中。“真”为居中，“假”为居左上。



例如：

```
影像框1.文件名 = "C:\WINDOWS\clock.avi"  
影像框1.居中播放 = 假  
影像框1.透明背景 = 真  
影像框1.播放 = 真  
影像框1.播放次数 = -1
```

程序执行后将“影像框1”当前播放的文件设为“C:\WINDOWS\clock.avi”；播放时设置影像位于影像框的左上；播放时透明影像框的背景；设置“影像框1”播放当前的影像；设置播放次数为循环播放。

### 3) “背景透明”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定播放影像时影像框背景是否透明。“真”为透明，“假”为不透明。本属性在播放多数影像时效果不是很明显，但建议设置为“真”。

例如：

```
影像框1.文件名 = "C:\WINDOWS\clock.avi"  
影像框1.居中播放 = 假  
影像框1.透明背景 = 真  
影像框1.播放 = 真  
影像框1.播放次数 = -1
```

程序执行后将“影像框1”当前播放的文件设为“C:\WINDOWS\clock.avi”；播放时设置影像位于影像框的左上；播放时透明影像框的背景；设置“影像框1”播放当前的影像；设置播放次数为循环播放。

### 4) “播放”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

如果指定了有效的影像文件，通过改变此属性可控制其播放或停止。如在设计时置本属性为“真”，运行时影像文件在影像框被创建后即自动开始播放。

例如：

```
影像框1.文件名 = "C:\WINDOWS\clock.avi"  
影像框1.居中播放 = 假  
影像框1.透明背景 = 真  
影像框1.播放 = 真  
影像框1.播放次数 = -1
```

程序执行后将“影像框1”当前播放的文件设为“C:\WINDOWS\clock.avi”；播放时设置影像位于影像框的左上；播放时透明影像框的背景；设置“影像框1”播放当前的影像；设置播放次数为循环播放。

### 5) “播放次数”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定影像的播放次数。如为 -1，则无限次地循环播放。

例如：

```
影像框1. 文件名 = "C:\WINDOWS\clock.avi"
影像框1. 居中播放 = 假
影像框1. 透明背景 = 真
影像框1. 播放 = 真
影像框1. 播放次数 = -1
```

程序执行后将“影像框1”当前播放的文件设为“C:\WINDOWS\clock.avi”；播放时设置影像位于影像框的左上；播放时透明影像框的背景；设置“影像框1”播放当前的影像；设置播放次数为循环播放。

## 4.12 数据库类组件

### 4.12.1 数据库提供者

数据库提供者用作为数据源提供对易语言数据库的存取操作。它使用数据库作为数据的存储仓库，不支持以下数据操作接口：1（置行高）；2（置类型）；3（置文本色）；4（置背景色）；5（置字体名）；6（置字体尺寸）；7（置字体属性）；8（置边距）；9（置文本输入格式）；10（置对齐方式）；11（置密码方式）；12（合并）；13（分解）；14（加线条）；15（删线条）；16（初始尺寸时同时改变列数）；17（在中间插入行）；18（插入列）；19（删除列）。

如果想对数据进行以上操作，应该先将数据通过数据源导出到通用提供者中。

数据库提供者属性

#### 1) “数据库文件名”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定与数据库提供者绑定的数据库文件名。

例如：

```
数据库提供者1. 数据库文件名 = "C:\MyeDb.edb"
数据库提供者1. 字节集字段处理 = 1
数据库提供者1. 数据库密码 = "eyuyan"
```

程序执行后将“数据库提供者1”与“C:\MyeDb.edb”易语言数据库文件绑定；指定将数据库中的字节集字段按照图片处理；设置数据库的访问密码为“eyuyan”。

#### 2) “字节集字段处理”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定对字节集类型字段的处理方式。可供选择的属性值有：0（跳过）；1（视为图片数据）；2（视为字节集数据）。

例如：

```
数据库提供者1.数据库文件名 = "C:\MyeDb.edb"  
数据库提供者1.字节集字段处理 = 1  
数据库提供者1.数据库密码 = "eyuyan"
```

程序执行后将“数据库提供者1”与“C:\MyeDb.edb”易语言数据库文件绑定；指定将数据库中的字节集字段按照图片处理；设置数据库的访问密码为“eyuyan”。

### 3) “数据库密码”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于提供访问数据库时所需要的密码。

例如：

```
数据库提供者1.数据库文件名 = "C:\MyeDb.edb"  
数据库提供者1.字节集字段处理 = 1  
数据库提供者1.数据库密码 = "eyuyan"
```

程序执行后将“数据库提供者1”与“C:\MyeDb.edb”易语言数据库文件绑定；指定将数据库中的字节集字段按照图片处理；设置数据库的访问密码为“eyuyan”。

## 4.12.2 数据源

数据源一般用于配合各种数据提供者窗口组件提供数据。数据源组件中内置了一系列对数据库的操作，可以简化窗口组件与数据库的关联操作。

数据源的另一个主要功能是和表格相连，进行各种报表操作。

数据源提供对各种来源数据的统一操作接口。数据源组件各个按钮的含义分别是：到首记录、到上一记录、到下一记录、到尾记录、添加新记录、删除当前记录。

### 4.12.2.1 数据源属性

#### 1) “数据提供者”属性

**属性类型：**文本型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定本数据源所基于的数据提供者组件名，所有数据存取操作都将通过数据提供者实际完成。

例如：

```
数据源_属性.数据提供者 = "数据提供者1"
```

程序执行后设置“数据源\_属性”的数据提供者“数据库提供者1”。

#### 2) “只读”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改





本属性用于指定是否允许操作者修改数据源内的数据，注意本属性并不限制在程序中直接修改数据。“真”为不允许修改，“假”为允许修改。

例如：

数据源\_属性.只读 = 真

程序执行后设置不允许修改“数据源\_属性”中的数据。

### 3) “允许添加”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许操作者添加记录到数据源内，注意本属性并不限制在程序中直接添加记录。“真”为允许，“假”为不允许。

例如：

数据源\_属性.允许添加 = 假

程序执行后设置不允许向“数据源\_属性”中添加数据。

### 4) “允许删除”属性

**属性类型：**逻辑型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改

本属性用于指定是否允许操作者删除数据源内的记录，注意本属性并不限制在程序中直接删除记录。“真”为允许，“假”为不允许。

例如：

数据源\_属性.允许删除 = 假

程序执行后设置不允许删除“数据源\_属性”中的数据。

## 4.12.2.2 数据源事件

### 1) “当前记录改变”事件

事件原型为：

**返回值类型：**无返回值；无参数

当操作者按下移动记录按钮移动了当前记录指针后，会产生本事件。

例如：

子程序名	返回值类型	公开	备注
_数据源1_当前记录改变			

» 输出调试文本 (数据源1.取记录号 ()

程序执行后，当“数据源1”当前记录指针被改变后在输出面板中显示当前所在的记录号。

### 2) “添加记录”事件

事件原型为：

**返回值类型：**无返回值；无参数



当操作者按下添加记录按钮添加了新记录后，会产生本事件。  
例如：

子程序名	返回值类型	公开	备注
数据源1_添加记录			

» 输出调试文本（“添加记录”）

程序执行后，当点击添加记录按钮后在输出面板中显示一段文字。

3) “删除记录”事件

事件原型为：

返回值类型：无返回值；无参数

当操作者按下删除记录按钮删除了当前记录后，会产生本事件。

例如：

子程序名	返回值类型	公开	备注
数据源1_删除记录			

» 输出调试文本（“删除记录”）

程序执行后，当点击删除按钮后在输出面板中显示一段文字。

4.12.2.3 数据源命令

1) “到首记录”命令

命令原型为：

<无返回值> 对象. 到首记录 ()

本命令用于将数据源中当前记录指针移动到第一条记录上。

例如：

数据源\_命令.到首记录 ()

程序执行后将“数据源\_命令”的当前记录指针指向第一条记录。

2) “到尾记录”命令

命令原型为：

<无返回值> 对象. 到尾记录 ()

本命令用于将数据源中当前记录指针移动到最后一条记录上。

例如：

数据源\_命令.到尾记录 ()

程序执行后将“数据源\_命令”的当前记录指针指向最后一条记录。

3) “跳过”命令

命令原型为：





<无返回值> 对象. 跳过 ([整数型 欲跳过的记录数])

本命令用于将数据源中当前记录指针向前或者向后移动数条记录。

参数<1>指定了欲跳过的记录数量。参数值如果为负数，则向前移动，否则向后移动。如果本参数被省略，默认值为1，即向后移动一条记录。

例如：

数据源\_命令.跳过(3)

程序执行后将“数据源\_命令”的记录指针跳到当前记录的向后3条处。

4) “跳到”命令

命令原型为：

<无返回值> 对象. 跳到 (整数型 欲跳到的记录号)

本命令用于改变数据源中的当前记录指针到指定的记录号。

参数<1>指定了欲跳转到的目标记录号。记录号从1开始，即首记录的记录号为1，依此类推。

例如：

数据源\_命令.跳到(5)

程序执行后跳到“数据源\_命令”的第5条记录。

5) “取记录号”命令

命令原型为：

<整数型> 对象. 取记录号 ()

本命令用于获取数据源当前记录指针所指向记录的记录号。记录号从1开始。如果当前记录指针无效，所返回记录号为0。

例如：

输出调试文本(数据源\_命令.取记录号())

程序执行后在输出面板中显示“数据源\_命令”记录指针所在的记录号。

6) “置表头行数”命令

命令原型为：

<无返回值> 对象. 置表头行数 ([整数型 表头行数])

本命令用于设置当以表格的形式表现数据源中的数据时表头所占的行数。表头行在表格中显示时不会滚动，在打印时会自动打印在每一页的顶部。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新的表头行数。参数值如果为-1，则自动调整由于单元格组合而产生的行格式错误。如果本参数被省略，默认值为-1。





例如:

数据源\_命令.置表头行数 (2)

输出调试文本 (数据源\_命令.取表头行数 ())

程序执行后设置“数据源\_命令”的表头行数为2行;在输出面板中显示“数据源\_命令”当前表头的行数。

7) “取表头行数”命令

命令原型为:

<整数型> 对象.取表头行数 ()

本命令用于获取当以表格的形式表现数据源中的数据时表头所占的行数。表头行在表格中显示时不会滚动,在打印时会自动打印在每一页的顶部。

例如:

数据源\_命令.置表头行数 (2)

输出调试文本 (数据源\_命令.取表头行数 ())

程序执行后设置“数据源\_命令”的表头行数为2行;在输出面板中显示“数据源\_命令”当前表头的行数。

8) “置表头列数”命令

命令原型为:

<无返回值> 对象.置表头列数 ([整数型 表头列数])

本命令用于设置当以表格的形式表现数据源中的数据时表头所占的列数。表头列在表格中显示时不会滚动。

**注解:** 如果数据源所使用的数据提供者不支持此特性,本命令将被忽略。

参数<1>指定了新的表头列数。参数值如果为-1,则自动调整由于单元格组合而产生的列格式错误。如果本参数被省略,默认值为-1。

例如:

数据源\_命令.置表头列数 (3)

输出调试文本 (数据源\_命令.取表头列数 ())

程序执行后设置“数据源\_命令”的表头列数为3列;在输出面板中显示“数据源\_命令”当前的表头列数。

9) “取表头列数”命令

命令原型为:

<整数型> 对象.取表头列数 ()

本命令用于获取当以表格的形式表现数据源中的数据时表头所占的列数。表头列在表格中显示时不会滚动。





例如：

数据源\_命令.置表头列数(3)

输出调试文本(数据源\_命令.取表头列数())

程序执行后设置“数据源\_命令”的表头列数为3列；在输出面板中显示“数据源\_命令”当前的表头列数。

#### 10) “置行高”命令

命令原型为：

<无返回值> 对象. 置行高 (整数型 行号, [整数型 行数], 整数型 高度)

本命令用于设置数据源中数据行在表现时的高度，单位为0.1毫米。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新行高应用范围的起始行号。行号从1开始。

参数<2>指定了应用范围的行数。如果本参数被省略，默认值为1。

参数<3>指定了新的行高。单位0.1毫米。

例如：

数据源\_命令.置行高(1, 2, 20)

输出调试文本(数据源\_命令.取行高(1))

程序执行后设置“数据源\_命令”的第一、第二行行高为2毫米；在输出面板中显示第一行的行高。

#### 11) “取行高”命令

命令原型为：

<整数型> 对象. 取行高 (整数型 行号)

本命令用于获取数据源中指定行在表现时的高度，单位为0.1毫米。

参数<1>指定了欲获得行高的行号。行号从1开始。

例如：

数据源\_命令.置行高(1, 2, 20)

输出调试文本(数据源\_命令.取行高(1))

程序执行后设置“数据源\_命令”的第一、第二行行高为2毫米；在输出面板中显示第一行的行高。

#### 12) “置列宽”命令

命令原型为：

<无返回值> 对象. 置列宽 (整数型 列号, [整数型 列数], 整数型 宽度)

本命令用于设置数据源中数据列在表现时的宽度，单位为0.1毫米。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新列宽应用范围的起始列号。列号从1开始。





参数<2>指定了应用范围的列数。如果本参数被省略，默认值为1。

参数<3>指定了新的列宽。单位0.1毫米。

例如：

数据源\_命令.置列宽 (2, 2, 30)

输出调试文本 (数据源\_命令.取列宽 (3))

程序执行后设置“数据源\_命令”第二、第三列的列宽为3毫米；在输出面板中显示第三列的列宽。

### 13) “取列宽”命令

命令原型为：

<整数型> 对象.取列宽 (整数型 列号)

本命令用于获取数据源中指定列在表现时的宽度，单位为0.1毫米。

参数<1>指定了欲获取宽度的列号。列号从1开始。

例如：

数据源\_命令.置列宽 (2, 2, 30)

输出调试文本 (数据源\_命令.取列宽 (3))

程序执行后设置“数据源\_命令”第二、第三列的列宽为3毫米；在输出面板中显示第三列的列宽。

### 14) “置类型”命令

命令原型为：

<无返回值> 对象.置类型 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 类型])

本命令用于设置数据源中指定单元格的数据类型。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新类型应用区域的起始行号。行号从1开始。

参数<2>指定了新类型应用区域的起始列号。列号从1开始。

参数<3>指定了应用区域的行数。如果本参数被省略，默认值为1。

参数<4>指定了应用区域的列数。如果本参数被省略，默认值为1。

参数<5>指定了新的类型值。可以为以下常量值之一：1（#文本）；2（#图片文件名）；3（#图片数据）；4（#字节集数据）。如果本参数被省略，默认值为“#文本”。

例如：

数据源\_命令.置类型 (1, 2, 2, 2, #图片数据)

输出调试文本 (数据源\_命令.取类型 (1, 3))

程序执行后设置“数据源\_命令”的第一行第二列起，两行，两列的数据类型都为“图片数据”；在输出面板中显示第一行第三列的单元格类型。





## 15) “取类型”命令

命令原型为:

&lt;整数型&gt; 对象. 取类型 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格的数据类型。

返回值为以下常量值之一： 1（#文本）； 2（#图片文件名）； 3（#图片数据）； 4（#字节集数据）。

参数&lt;1&gt;指定了欲取类型的单元格所在行号。行号从1开始。

参数&lt;2&gt;指定了欲取类型的单元格所在列号。列号从1开始。

例如:

数据源\_命令.置类型 (1, 2, 2, 2, #图片数据)

输出调试文本 (数据源\_命令.取类型 (1, 3))

程序执行后设置“数据源\_命令”的第一行第二列起，两行，两列的数据类型都为“图片数据”；在输出面板中显示第一行第三列的单元格类型。

## 16) “置文本颜色”命令

命令原型为:

<无返回值> 对象. 置文本色 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 文本颜色])

本命令用于设置数据源中指定单元格在表现时的文本颜色。

**注解:** 如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数&lt;1&gt;指定了新颜色应用区域起始行号。行号从1开始。

参数&lt;2&gt;指定了新颜色应用区域起始列号。列号从1开始。

参数&lt;3&gt;指定了新颜色应用区域的行数。如果本参数被省略，默认值为1。

参数&lt;4&gt;指定了新颜色应用区域的列数。如果本参数被省略，默认值为1。

参数<5>指定了新颜色的颜色值。如果本参数被省略，默认值为黑色。颜色值可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如:

数据源\_命令.置文本色 (1, 1, 1, 1, #蓝色)

输出调试文本 (数据源\_命令.取文本色 (1, 1))

程序执行后设置“数据源\_命令”中第一行第一列的单元格中文本的颜色为蓝色；在输出面板显示“数据源\_命令”中第一行第一列的单元格的文本颜色值。

## 17) “取文本色”命令

命令原型为:

&lt;整数型&gt; 对象. 取文本色 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现时的文本颜色。





参数<1>指定了欲取颜色单元格的所在行号。行号从1开始。

参数<2>指定了欲取颜色单元格的所在列号。列号从1开始。

例如：

```
数据源_命令.置文本色(1, 1, 1, 1, #蓝色)
```

```
输出调试文本(数据源_命令.取文本色(1, 1))
```

程序执行后设置“数据源\_命令”中第一行第一列的单元格中文本的颜色为蓝色；在输出面板显示“数据源\_命令”中第一行第一列的单元格的文本颜色值。

#### 18) “置背景色”命令

命令原型为：

```
<无返回值> 对象.置背景色 (整数型 行号, 整数型 列号, [整数型 行数],  
[整数型 列数], [整数型 背景颜色])
```

本命令用于设置数据源中指定单元格在表现时的背景颜色。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新背景颜色应用范围的起始行号。行号从1开始。

参数<2>指定了新背景颜色应用范围的起始列号。列号从1开始。

参数<3>指定了新背景颜色应用范围的行数。如果本参数被省略，默认值为1。

参数<4>指定了新背景颜色应用范围的列数。如果本参数被省略，默认值为1。

参数<5>指定了新背景颜色值。如果本参数被省略，默认值为透明色。颜色值可以通过“取颜色值”命令调配，或使用易语言中内置的颜色常量如“#红色”等。

例如：

```
数据源_命令.置背景色(1, 1, 1, 1, #红色)
```

```
输出调试文本(数据源_命令.取背景色(1, 1))
```

程序执行后设置“数据源\_命令”中第一行第一列的单元格背景颜色为红色；在输出面板中显示“数据源\_命令”中第一行第一列的单元格背景颜色值。

#### 19) “取背景色”命令

命令原型为：

```
<整数型> 对象.取背景色 (整数型 行号, 整数型 列号)
```

本命令用于获取数据源中指定单元格在表现时的背景颜色。

参数<1>指定了欲获取颜色值的单元格所在行号。行号从1开始。

参数<2>指定了欲获取颜色值的单元格所在列号。列号从1开始。

例如：

```
数据源_命令.置背景色(1, 1, 1, 1, #红色)
```

```
输出调试文本(数据源_命令.取背景色(1, 1))
```





程序执行后设置“数据源\_命令”中第一行第一列的单元格背景颜色为红色；在输出面板中显示“数据源\_命令”中第一行第一列的单元格背景颜色值。

## 20) “置字体名”命令

命令原型为：

<无返回值> 对象. 置字体名 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [文本型 字体名称])

本命令用于设置数据源中指定单元格在表现时所使用字体的名称。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新字体的应用范围起始行号。行号从1开始。

参数<2>指定了新字体的应用范围起始列号。列号从1开始。

参数<3>指定了新字体的应用范围行数。如果本参数被省略，默认值为1。

参数<4>指定了新字体的应用范围列数。如果本参数被省略，默认值为1。

参数<5>指定了新字体的名称。如果本参数被省略，默认值为“宋体”。

例如：

数据源\_命令.置字体名(1, 1, 1, 1, “黑体”)

输出调试文本(数据源\_命令.取字体名(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列的单元格字体为“黑体”；在输出面板中显示“数据源\_命令”中第一行第一列的单元格字体名称。

## 21) “取字体名”命令

命令原型为：

<文本型> 对象. 取字体名 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现时所使用字体的名称。

参数<1>指定了欲获取字体名称的单元格所在行号。行号从1开始。

参数<2>指定了欲获取字体名称的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置字体名(1, 1, 1, 1, “黑体”)

输出调试文本(数据源\_命令.取字体名(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列的单元格字体为“黑体”；在输出面板中显示“数据源\_命令”中第一行第一列的单元格字体名称。

## 22) “置字体尺寸”命令

命令原型为：

<无返回值> 对象. 置字体尺寸 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 字体尺寸])

本命令用于设置数据源中指定单元格在表现时所使用字体的尺寸，单位为0.1毫米。





**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新字体尺寸应用范围的起始行。行号从1开始。

参数<2>指定了新字体尺寸应用范围的起始列。列号从1开始。

参数<3>指定了新字体应用范围的行数。如果本参数被省略，默认值为1。

参数<4>指定了新字体应用范围的列数。如果本参数被省略，默认值为1。

参数<5>指定了新的字体尺寸。如果本参数被省略，默认值为40。

例如：

数据源\_命令.置字体尺寸(1, 1, 1, 1, 30)

输出调试文本(数据源\_命令.取字体尺寸(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格中的文字的尺寸为3毫米；在输出面板中显示“数据源\_命令”中第一行第一列单元格中文字的尺寸。

### 23) “取字体尺寸”命令

命令原型为：

<整数型> 对象.取字体尺寸(整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现时所使用字体的尺寸，单位为0.1毫米。

参数<1>指定了欲取字体尺寸的单元格所在行号。行号从1开始。

参数<2>指定了欲取字体尺寸的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置字体尺寸(1, 1, 1, 1, 30)

输出调试文本(数据源\_命令.取字体尺寸(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格中的文字的尺寸为3毫米；在输出面板中显示“数据源\_命令”中第一行第一列单元格中文字的尺寸。

### 24) “置字体属性”命令

命令原型为：

<无返回值> 对象.置字体属性(整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 字体属性])

本命令用于设置数据源中指定单元格在表现时所使用字体的属性。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新字体效果应用范围的起始行号。行号从1开始。

参数<2>指定了新字体效果应用范围的起始列号。列号从1开始。

参数<3>指定了新字体效果应用范围的行数。如果本参数被省略，默认值为1。

参数<4>指定了新字体效果应用范围的列数。如果本参数被省略，默认值为1。

参数<5>指定了新效果的值或效果组合后的值。





可以为以下常量值之一或之和：1（#粗体）；2（#斜体）；4（#下划线）；8（#删除线）。如果本参数被省略，默认值为0。

例如：

```
数据源_命令.置字体属性(1, 1, 1, 1, #粗体)
```

```
输出调试文本(数据源_命令.取字体属性(1, 1))
```

程序执行后设置“数据源\_命令”的第一行第一列单元格中的文字拥有“粗体”效果；在输出面板中显示“数据源\_命令”中第一行第一列单元格中文字拥有的字体属性。

## 25) “取字体属性”命令

命令原型为：

```
<整数型> 对象. 取字体属性 (整数型 行号, 整数型 列号)
```

本命令用于获取数据源中指定单元格在表现时所使用字体的属性。返回值为以下常量值之一或之和：1（#粗体）；2（#斜体）；4（#下划线）；8（#删除线）。

参数<1>指定了欲获取字体属性的单元格所在行号。行号从1开始。

参数<2>指定了欲获取字体属性的单元格所在列号。列号从1开始。

例如：

```
数据源_命令.置字体属性(1, 1, 1, 1, #粗体)
```

```
输出调试文本(数据源_命令.取字体属性(1, 1))
```

程序执行后设置“数据源\_命令”的第一行第一列单元格中的文字拥有“粗体”效果；在输出面板中显示“数据源\_命令”中第一行第一列单元格中文字拥有的字体属性。

## 26) “置边距”命令

命令原型为：

```
<无返回值> 对象. 置边距 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 边距])
```

本命令用于设置数据源中指定单元格在表现其中数据时至其单元格边框之间的空白距离，单位为0.1毫米。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新的边距应用范围起始行号。行号从1开始。

参数<2>指定了新的边距应用范围起始列号。列号从1开始。

参数<3>指定了新的边距应用范围行数。如果本参数被省略，默认值为1。

参数<4>指定了新的边距应用范围列数。如果本参数被省略，默认值为1。

参数<5>指定了新的边距值。如果本参数被省略，默认值为9。

例如：

```
数据源_命令.置边距(1, 1, 1, 1, 10)
```

```
输出调试文本(数据源_命令.取边距(1, 1))
```





程序执行后设置“数据源\_命令”中的第一行第一列的单元格与文字的间距设为1毫米；在输出面板中显示“数据源\_命令”中的第一行第一列的单元格与文字间的间距。

#### 27) “取边距”命令

命令原型为：

<整数型> 对象. 取边距 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现其中数据时至其单元格边框之间的空白距离，单位为0.1毫米。

参数<1>指定了欲获取边距的单元格所在行号。行号从1开始。

参数<2>指定了欲获取边距的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置边距 (1, 1, 1, 1, 10)

输出调试文本 (数据源\_命令.取边距 (1, 1))

程序执行后设置“数据源\_命令”中的第一行第一列的单元格与文字的间距设为1毫米；在输出面板中显示“数据源\_命令”中的第一行第一列的单元格与文字间的间距。

#### 28) “置文本输入格式”命令

命令原型为：

<无返回值> 对象. 置文本输入格式 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 输入格式])

本命令用于设置数据源中某文本型单元格的输入格式。当新的数据通过输入方式更新到该单元格内之前，将首先自动根据此格式转换。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新的输入格式的应用范围起始行号。行号从1开始。

参数<2>指定了新的输入格式的应用范围起始列号。列号从1开始。

参数<3>指定了新的输入格式的应用范围行数。如果本参数被省略，默认值为1。

参数<4>指定了新的输入格式的应用范围列数。如果本参数被省略，默认值为1。

参数<5>指定了新的输入格式值。

可以为以下常量值之一：0（#通常型）；1（#字节型）；2（#短整数型）；3（#整数型）；4（#长整数型）；5（#小数型）；6（#双精度小数型）；7（#逻辑型）；8（#日期时间型）。如果本参数被省略，默认值为“#通常型”。

例如：

数据源\_命令.置文本输入格式 (1, 1, 1, 1, #日期时间型)

输出调试文本 (数据源\_命令.取文本输入格式 (1, 1))

程序执行后设置“数据源\_命令”第一行第一列的单元格使用日期时间型输入格式；在输出面板中显示“数据源\_命令”第一行第一列的单元格当前的输入格式类型值。





## 29) “取文本输入格式” 命令

命令原型为:

<整数型> 对象. 取文本输入格式 (整数型 行号, 整数型 列号)

本命令用于获取数据源中某文本型单元格的输入格式。当新的数据通过输入方式更新到该单元格内之前, 将首先自动根据此格式转换。

返回值为以下常量值之一: 0 (#通常型); 1 (#字节型); 2 (#短整数型); 3 (#整数型); 4 (#长整数型); 5 (#小数型); 6 (#双精度小数型); 7 (#逻辑型); 8 (#日期时间型)。

参数<1>指定了欲获取输入格式的单元格所在行号。行号从1开始。

参数<2>指定了欲获取输入格式的单元格所在列号。列号从1开始。

例如:

数据源\_命令.置文本输入格式(1, 1, 1, 1, #日期时间型)

输出调试文本(数据源\_命令.取文本输入格式(1, 1))

程序执行后设置“数据源\_命令”第一行第一列的单元格使用日期时间型输入格式; 在输出面板中显示“数据源\_命令”第一行第一列的单元格当前的输入格式类型值。

## 30) “置对齐方式” 命令

命令原型为:

<无返回值> 对象. 置对齐方式 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 对齐方式])

本命令用于设置数据源中指定单元格在表现时所使用的对齐方式。

**注解:** 如果数据源所使用的数据提供者不支持某些属性, 该属性将被忽略。

参数<1>指定了新的对齐方式应用范围的起始行号。行号从1开始。

参数<2>指定了新的对齐方式应用范围的起始列号。列号从1开始。

参数<3>指定了新的对齐方式应用范围的行数。如果本参数被省略, 默认值为1。

参数<4>指定了新的对齐方式应用范围的列数。如果本参数被省略, 默认值为1。

参数<5>指定了新的对齐方式样式值。

可以为以下常量值之一: 1 (#上左); 2 (#上中); 3 (#上右); 4 (#中左); 5 (#中中); 6 (#中右); 7 (#下左); 8 (#下中); 9 (#下右); 10 (#缩放图片); 11 (#居中图片); 12 (#缩放居中图片); 13 (#平铺图片); 14 (#缩放平铺图片)。其中10到14对齐方式仅对图片型单元格有效。如果本参数被省略, 默认值为“#上左”。

例如:

数据源\_命令.置对齐方式(1, 1, 1, 1, #中中)

输出调试文本(数据源\_命令.取对齐方式(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格的对齐方式为纵向、横向全





居中样式；在输出面板中显示“数据源\_命令”中第一行第一列单元格的对齐方式值。

### 31) “取对齐方式” 命令

命令原型为：

<整数型> 对象. 取对齐方式 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现时所使用的对齐方式。

返回值为以下常量值之一：1（#上左）；2（#上中）；3（#上右）；4（#中左）；5（#中中）；6（#中右）；7（#下左）；8（#下中）；9（#下右）；10（#缩放图片）；11（#居中图片）；12（#缩放居中图片）；13（#平铺图片）；14（#缩放平铺图片）。

参数<1>指定了欲获取对齐方式的单元格所在行号。行号从1开始。

参数<2>指定了欲获取对齐方式的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置对齐方式(1, 1, 1, 1, #中中)

输出调试文本(数据源\_命令.取对齐方式(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格的对齐方式为纵向、横向全居中样式；在输出面板中显示“数据源\_命令”中第一行第一列单元格的对齐方式值。

### 32) “置密码方式” 命令

命令原型为：

<无返回值> 对象. 置密码方式 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], 逻辑型 设置值)

本命令用于设置数据源中指定单元格在表现时是否使用密码显示样式。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了新方式的应用范围起始行号。行号从1开始。

参数<2>指定了新方式的应用范围起始列号。列号从1开始。

参数<3>指定了新方式的应用范围行数。如果本参数被省略，默认值为1。

参数<4>指定了新方式的应用范围列数。如果本参数被省略，默认值为1。

参数<5>指定了是否使用密码样式。如设置值为“真”，则数据源中指定单元格在表现时以密码字符呈现，为“假”则不使用密码样式。

例如：

数据源\_命令.置密码方式(1, 1, 1, 1, 真)

输出调试文本(数据源\_命令.取密码方式(1, 1))

程序执行后设置“数据源\_命令”中的第一行第一列单元格使用密码显示方式；在输出面板中显示“数据源\_命令”中第一行第一列的单元格是否使用密码显示方式。

### 33) “取密码方式” 命令

命令原型为：





**<逻辑型> 对象. 取密码方式 (整数型 行号, 整数型 列号)**

本命令用于获取数据源中指定单元格在表现时是否使用密码方式。返回“真”表示使用，“假”表示没使用。

参数<1>指定了欲获取是否使用密码样式的单元格所在行号。行号从1开始。

参数<2>指定了欲获取是否使用密码样式的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置密码方式(1, 1, 1, 1, 真)

输出调试文本(数据源\_命令.取密码方式(1, 1))

程序执行后设置“数据源\_命令”中的第一行第一列单元格使用密码显示方式；在输出面板中显示“数据源\_命令”中第一行第一列的单元格是否使用密码显示方式。

#### 34) “置文本”命令

命令原型为：

**<无返回值> 对象. 置文本 (整数型 行号, 整数型 列号, 文本型 文本或图片文件名)**

本命令用于设置数据源中指定单元格的文本内容。

**注解：**该单元格类型必须为“#文本”或“#图片文件名”。

参数<1>指定了欲设置内容单元格的所在行号。行号从1开始。

参数<2>指定了欲设置内容单元格的所在列号。列号从1开始。

参数<3>指定了欲设置的新内容。

例如：

数据源\_命令.置文本(1, 1, “易语言”)

输出调试文本(数据源\_命令.取文本(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格的内容为“易语言”；在输出面板中显示“数据源\_命令”中第一行第一列单元格中的文本。

#### 35) “取文本”命令

命令原型为：

**<文本型> 对象. 取文本 (整数型 行号, 整数型 列号)**

本命令用于获取数据源中指定单元格的文本内容。

参数<1>指定了欲获取文本的单元格所在行号。行号从1开始。

参数<2>指定了欲获取文本的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置文本(1, 1, “易语言”)

输出调试文本(数据源\_命令.取文本(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列单元格的内容为“易语言”；在输





出面板中显示“数据源\_命令”中第一行第一列单元格中的文本。

36) “置数据” 命令

命令原型为：

<无返回值> 对象. 置数据 (整数型 行号, 整数型 列号, 字节集 数据)

本命令用于设置数据源中指定单元格的图片或字节集数据内容。

**注解：**该单元格类型必须为“#图片数据”或“#字节集数据”。

参数<1>指定了欲设置新数据的单元格所在行号。行号从1开始。

参数<2>指定了欲设置新数据的单元格所在列号。列号从1开始。

参数<3>指定了欲设置的新数据。

例如：

变量名	类 型	静态	数组	备 注
读入的数据	字节集			
取得的数据	字节集			

读入的数据 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
 数据源1.置数据 (1, 1, 读入的数据)  
 取得的数据 = 数据源1.取数据 (1, 1)  
 写到文件 (“C:\test.bmp”, 取得的数据)

程序执行后将由“C:\WINDOWS\Prairie Wind.bmp”读取的数据保存到“数据源1”中第一行第一列的单元格中；获取“数据源1”中第一行第一列的单元格中的数据保存为“C:\text.bmp”。

37) “取数据” 命令

命令原型为：

<字节集> 对象. 取数据 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格的图片或字节集数据内容。

参数<1>指定了欲获取数据单元格所在的行号。行号从1开始。

参数<2>指定了欲获取数据单元格所在的列号。列号从1开始。

例如：

变量名	类 型	静态	数组	备 注
读入的数据	字节集			
取得的数据	字节集			

读入的数据 = 读入文件 (“C:\WINDOWS\Prairie Wind.bmp”)  
 数据源1.置数据 (1, 1, 读入的数据)  
 取得的数据 = 数据源1.取数据 (1, 1)  
 写到文件 (“C:\test.bmp”, 取得的数据)

程序执行后将由“C:\WINDOWS\Prairie Wind.bmp”读取的数据保存到“数据源1”





中第一行第一列的单元格中；获取“数据源1”中第一行第一列的单元格中的数据保存为“C:\text.bmp”。

### 38) “合并”命令

命令原型为：

<无返回值> 对象. 合并 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数])

本命令用于组合数据源中指定范围内的单元格，使之以一个单元格的形式表现。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了欲合并范围的起始行号。行号从1开始。

参数<2>指定了欲合并范围的起始列号。列号从1开始。

参数<3>指定了欲合并范围的行数。如果本参数被省略，默认值为1。

参数<4>指定了欲合并范围的列数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.合并 (1, 1, 2, 2)

程序执行后将合并“数据源\_命令”中 (1, 1) 到 (2, 2) 范围的单元格为一个单元格。

### 39) “分解”命令

命令原型为：

<无返回值> 对象. 分解 (整数型 行号, 整数型 列号)

本命令用于分解数据源中指定的已经组合的单元格，行列参数指向被组合单元格内的任何一个单元格即可。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了欲分解的单元格所在行号。行号从1开始。

参数<2>指定了欲分解的单元格所在列号。列号从1开始。

例如：

如果真 (数据源1.是否被合并 (1, 1, #右单元格))  
数据源1.分解 (1, 1)

程序执行后判断“数据源1”中第一行第一列单元格是否和其右边单元格有合并，如果有则分解它们。

### 40) “是否被合并”命令

命令原型为：

<逻辑型> 对象. 是否被合并 (整数型 行号, 整数型 列号, 整数型 周边单元格)

本命令用于判断指定单元格是否和周围单元格组合过。如果数据源中指定单元格与周边指定单元格有组合关系，返回“真”，否则返回“假”。



参数<1>指定了欲测试的单元格所在行号。行号从1开始。

参数<2>指定了欲测试的单元格所在列号。列号从1开始。

参数<3>指定了欲测试的周边单元格。

可以为以下常量值之一：1（#左单元格）；2（#上单元格）；3（#右单元格）；4（#下单元格）。

例如：

```

如果真 (数据源1.是否被合并 (1, 1, #右单元格))
    数据源1.分解 (1, 1)
    
```

程序执行后判断“数据源1”中第一行第一列单元格是否和其右边单元格有合并，如果有则分解它们。

#### 41) “加线条”命令

命令原型为：

```

<无返回值> 对象.加线条 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 线条类型])
    
```

本命令用于为数据源中指定范围内单元格添加线条。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了欲添加线条的单元格范围起始行号。行号从1开始。

参数<2>指定了欲添加线条的单元格范围起始列号。列号从1开始。

参数<3>指定了欲添加线条的单元格范围行数。如果本参数被省略，默认值为1。

参数<4>指定了欲添加线条的单元格范围列数。如果本参数被省略，默认值为1。

参数<5>指定了欲添加线条的类型。

可以为以下常量值之一或之和：1（#左边框）；2（#上边框）；4（#右边框）；8（#下边框）；16（#水平线）；32（#垂直线）；64（#单斜线）；128（#双斜线）；256（#交叉线）。如果本参数被省略，默认值为“#左边框 + #上边框 + #右边框 + #下边框”。

例如：

```

数据源_命令.加线条 (1, 1, 1, 1, #单斜线)
    
```

程序执行后为“数据源\_命令”中第一行第一列的单元格添加一条单斜线。

#### 42) “删除线”命令

命令原型为：

```

<无返回值> 对象.删线条 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], [整数型 线条类型])
    
```

本命令用于将数据源中指定范围内单元格内的线条去除。

**注解：**如果数据源所使用的数据提供者不支持此特性，本命令将被忽略。





参数<1>指定了欲删除线条的单元格范围起始行号。行号从1开始。

参数<2>指定了欲删除线条的单元格范围起始列号。列号从1开始。

参数<3>指定了欲删除线条的单元格范围行数。如果本参数被省略，默认值为1。

参数<4>指定了欲删除线条的单元格范围列数。如果本参数被省略，默认值为1。

参数<5>指定了欲删除线条的类型。

可以为以下常量值之一或之和：1（#左边框）；2（#上边框）；4（#右边框）；8（#下边框）；16（#水平线）；32（#垂直线）；64（#单斜线）；128（#双斜线）；256（#交叉线）。如果本参数被省略，默认值为“#左边框 + #上边框 + #右边框 + #下边框”。

例如：

```
-- 如果真 (数据源1. 是否有线条 (1, 1, #单斜线) = 真)
数据源1. 删线条 (1, 1, 1, 1, #单斜线)
```

程序执行后如果“数据源1”中第一行第一列单元格有“单斜线”样式，则删除它的“单斜线”样式。

#### 43) “是否有线条”命令

命令原型为：

<逻辑型> 对象. 是否有线条 (整数型 行号, 整数型 列号, 整数型 线条类型)

本命令用于判断指定数据源中指定单元格是否添加了线条。如果数据源中指定单元格添加了指定线条，返回“真”，否则返回“假”。

参数<1>指定了欲测试单元格所在的行号。行号从1开始。

参数<2>指定了欲测试单元格所在的列号。列号从1开始。

参数<3>指定了欲测试线条的类型。

可以为以下常量值之一或之和：1（#左边框）；2（#上边框）；4（#右边框）；8（#下边框）；64（#单斜线）；128（#双斜线）；256（#交叉线）。

例如：

```
-- 如果真 (数据源1. 是否有线条 (1, 1, #单斜线) = 真)
数据源1. 删线条 (1, 1, 1, 1, #单斜线)
```

程序执行后如果“数据源1”中第一行第一列单元格有“单斜线”样式，则删除它的“单斜线”样式。

#### 44) “清除”命令

命令原型为：

<无返回值> 对象. 清除 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数])

本命令用于清除数据源中指定范围内单元格内容为空文本。





参数<1>指定了欲清除的单元格范围起始行号。行号从1开始。

参数<2>指定了欲清除的单元格范围起始列号。列号从1开始。

参数<3>指定了欲清除的单元格范围行数。如果本参数被省略，默认值为1。

参数<4>指定了欲清除的单元格范围列数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.清除 (1, 1, 2, 2)

程序执行后会将“数据源\_命令”中 (1, 1) 到 (2, 2) 的单元格中的内容清空。

#### 45) “置初始属性”命令

命令原型为：

<无返回值> 对象.置初始属性 ([整数型 类型], [整数型 文本颜色], [整数型 背景颜色], [文本型 字体名称], [整数型 字体尺寸], [整数型 字体属性], [整数型 边距], [整数型 对齐方式], [整数型 输入格式], [逻辑型 添加边框线])

本命令用于设置在数据源中初始或者添加新单元格时默认的单元格初始属性。

参数<1>指定了初始单元格的类型。

可以为以下常量值之一：1 (#文本)；2 (#图片文件名)；3 (#图片数据)；4 (#字节集数据)。如果本参数被省略，默认值为“#文本”。

参数<2>指定了初始单元格中文字的颜色。如果本参数被省略，默认值为黑色。

参数<3>指定了初始单元格中文字的背景颜色。如果本参数被省略，默认值为透明色。

参数<4>指定了初始单元格中文字的字体名称。如果本参数被省略，默认值为“宋体”。

参数<5>指定了初始单元格中文字的大小。如果本参数被省略，默认值为40。

参数<6>指定了初始单元格中文字的显示效果。

可以为以下常量值之一或之和：1 (#粗体)；2 (#斜体)；4 (#下划线)；8 (#删除线)。如果本参数被省略，默认值为0。

参数<7>指定了初始单元格中文字的边距。如果本参数被省略，默认值为9。

参数<8>指定了初始单元格中文字的对齐方式。

可以为以下常量值之一：1 (#上左)；2 (#上中)；3 (#上右)；4 (#中左)；5 (#中中)；6 (#中右)；7 (#下左)；8 (#下中)；9 (#下右)；10 (#缩放图片)；11 (#居中图片)；12 (#缩放居中图片)；13 (#平铺图片)；14 (#缩放平铺图片)。其中 10 到 14 对齐方式仅对图片型单元格有效。如果本参数被省略，默认值为“#上左”。

参数<9>指定了初始单元格的内容输入方式。

可以为以下常量值之一：0 (#通常型)；1 (#字节型)；2 (#短整数型)；3 (#整数型)；4 (#长整数型)；5 (#小数型)；6 (#双精度小数型)；7 (#逻辑型)；8 (#日期





时间型)。如果本参数被省略,默认值为“#通常型”。

参数<10>指定了是否在单元格周围添加边框线。“真”为添加,“假”为不添加。如果本参数被省略,默认值为“假”。

例如:

数据源\_命令.置初始属性(#文本,#绿色,#黑色,“黑体”,35",#粗体,10,#中中,#通常型,真)

程序执行后设置“数据源\_命令”中的新单元格初始属性为新的值。

#### 46) “初始尺寸”命令

命令原型为:

<逻辑型> 对象. 初始尺寸 ([整数型 行数], [整数型 列数])

本命令用于初始数据源中数据的行列数,数据源中所有原有数据将被清除。

**注解:** 某些数据提供者可能不支持此命令。成功返回“真”,失败返回“假”。

参数<1>指定了初始拥有的行数。如果本参数被省略,默认值为1。

参数<2>指定了初始拥有的列数。如果本参数被省略,默认值为1。

例如:

数据源\_命令.初始尺寸(10,10)

程序执行后将“数据源\_命令”初始为10行10列。

#### 47) “取行数”命令

命令原型为:

<整数型> 对象. 取行数 ()

本命令用于获取数据源中现行数据行数。

例如:

输出调试文本(数据源\_命令.取行数())

程序执行后在输出面板中显示当前“数据源\_命令”的行数。

#### 48) “取列数”命令

命令原型为:

<整数型> 对象. 取列数 ()

本命令用于获取数据源中现行数据列数。

例如:

输出调试文本(数据源\_命令.取列数())

程序执行后在输出面板中显示当前“数据源\_命令”的列数。

#### 49) “插入行”命令

命令原型为:

<逻辑型> 对象. 插入行 (整数型 行号, [整数型 行数])





本命令用于在数据源中指定位置处插入新数据行。

**注解：**某些数据提供者可能不支持在中间插入行。成功返回“真”，失败返回“假”。

参数<1>指定在哪一行前插入新行。行号从1开始。

参数<2>指定了欲插入的行数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.插入行(1,5)

程序执行后在第一行前新插入5行。

#### 50) “添加行”命令

命令原型为：

<逻辑型> 对象.添加行 ([整数型 行数])

本命令用于在数据源中最后一行后插入新数据行。成功返回“真”，失败返回“假”。

参数<1>指定了欲添加的新行数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.添加行(5)

程序执行后向“数据源\_命令”中添加5行。

#### 51) “删除行”命令

命令原型为：

<逻辑型> 对象.删除行 (整数型 行号, [整数型 行数])

本命令用于在数据源中指定位置处删除数据行。成功返回“真”，失败返回“假”。

参数<1>指定了欲删除范围的起始行号。行号从1开始。

参数<2>指定了欲删除的行数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.删除行(2,5)

程序执行后删除“数据源\_命令”中第二行起的5行。

#### 52) “插入列”命令

命令原型为：

<逻辑型> 对象.插入列 (整数型 列号, [整数型 列数])

本命令用于在数据源中指定位置处插入新数据列。成功返回“真”，失败返回“假”。

参数<1>指定了新列的插入位置。列号从1开始。

参数<2>指定了欲插入的列数。如果本参数被省略，默认值为1。

例如：

数据源\_命令.插入列(1,3)

程序执行后在“数据源\_命令”中第一列前插入3个新列。





## 53) “删除列”命令

命令原型为:

&lt;逻辑型&gt; 对象. 删除列 (整数型 列号, [整数型 列数])

本命令用于在数据源中指定位置处删除数据列。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定了欲删除范围的起始列号。列号从1开始。

参数&lt;2&gt;指定了欲删除的列数。如果本参数被省略，默认值为1。

例如:

数据源\_命令.删除列 (2, 3)

程序执行后删除“数据源\_命令”中第二列起的3列。

## 54) “添加”命令

命令原型为:

<逻辑型> 对象. 添加 (通用型 数据源或数据提供者, [整数型 首行行号],  
[整数型 欲添加行数])

本命令用于将指定其他数据源或者数据提供者中的数据添加到本数据源的尾部。成功返回“真”，失败返回“假”。

参数&lt;1&gt;指定欲添加其数据到数据源中的其他数据源或数据提供者。

参数&lt;2&gt;指定了欲添加数据首行的行号。如果本参数被省略，默认值为1。

参数&lt;3&gt;指定了欲添加数据行的行数。如果参数值为-1，则添加首行后的所有数据行。如果本参数被省略，默认值为-1。

例如:

数据源\_命令.添加 (数据源2, 2, 5)

程序执行后将“数据源2”中的第二条记录起的5条记录添加到“数据源\_命令”中。

## 55) “存到字节集”命令

命令原型为:

&lt;字节集&gt; 对象. 存到字节集 ()

本命令用于将数据源中所有现有数据保存在字节集中返回。如果失败，将返回空字节集。

例如:

变量名	类型	静态	数组	备注
数据	字节集			

数据 = 数据源2.存到字节集 ()

数据源1.从字节集读 (数据)

程序执行后将“数据源2”中的数据拷贝到“数据源1”中。





## 56) “从字节集读”命令

命令原型为:

<逻辑型> 对象. 从字节集读 (字节集 数据源数据)

本命令用于将指定字节集内的数据源数据写入到数据源中, 数据源内所有原有数据将被清除。成功返回“真”, 失败返回“假”。

参数<1>指定了欲读取的字节集数据。

例如:

变量名	类型	静态	数组	备注
数据	字节集			

数据 = 数据源2. 存到字节集 ()  
数据源1. 从字节集读 (数据)

程序执行后将“数据源2”中的数据拷贝到“数据源1”中。

## 57) “存到文件”命令

命令原型为:

<逻辑型> 对象. 存到文件 (文本型 数据源数据文件名)

本命令用于将数据源中所有现有数据保存到指定文件中。成功返回“真”, 失败返回“假”。

参数<1>指定了存储数据文件的文件名。

例如:

数据源\_命令. 存到文件 (“c:\MyBak.bak”)

程序执行后将“数据源\_命令”中的数据保存到“c:\MyBak.bak”中。

## 58) “从文件读”命令

命令原型为:

<逻辑型> 对象. 从文件读 (文本型 数据源数据文件名)

本命令用于将指定文件内的数据源数据写入到数据源中, 数据源内所有原有数据将被清除。成功返回“真”, 失败返回“假”。

参数<1>指定了存储数据文件的文件名。

例如:

数据源\_命令. 从文件读 (“c:\MyBak.bak”)

程序执行后“数据源\_命令”从“c:\MyBak.bak”中恢复以前保存的数据。

## 59) “单元格到字节集”命令

命令原型为:

<字节集> 对象. 单元格到字节集 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数])





本命令用于将数据源中指定范围内单元格数据保存在字节集中返回。如果失败，将返回空字节集。

参数<1>指定了欲转换的区域起始行号。行号从1开始。

参数<2>指定了欲转换的区域起始列号。列号从1开始。

参数<3>指定了欲转换区域的行数。如果本参数被省略，默认值为1。

参数<4>指定了欲转换区域的列数。如果本参数被省略，默认值为1。

例如：

变量名	类型	静态	数组	备注
数据	字节集			

数据 = 数据源2. 单元格到字节集 (1, 1, 2, 2)

数据源1. 字节集到单元格 (2, 2, 数据, 真)

程序执行后将“数据源2”中(1, 1)到(2, 2)的单元格数据拷贝到“数据源1”的(2, 2)处。

#### 60) “字节集到单元格”命令

命令原型为：

<逻辑型> 对象. 字节集到单元格 (整数型 行号, 整数型 列号, 字节集 单元格数据, [逻辑型 自动扩展])

本命令用于将指定字节集内的单元格数据写入到数据源中指定位置。成功返回真，失败返回假。

参数<1>指定了将读入的数据置入的起始行号。行号从1开始。

参数<2>指定了将读入的数据置入的起始列号。列号从1开始。

参数<3>指定了欲置入的数据。

参数<4>指定如果数据源内的数据行列数无法容纳即将读入的数据，是否自动扩展。“真”为自动扩展，“假”为不自动扩展。如果本参数被省略，默认值为“真”。

例如：

变量名	类型	静态	数组	备注
数据	字节集			

数据 = 数据源2. 单元格到字节集 (1, 1, 2, 2)

数据源1. 字节集到单元格 (2, 2, 数据, 真)

程序执行后将“数据源2”中(1, 1)到(2, 2)的单元格数据拷贝到“数据源1”的(2, 2)处。

#### 61) “单元格到文件”命令

命令原型为：





<逻辑型> 对象. 单元格到文件 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], 文本型 数据源数据文件名)

本命令用于将数据源中指定范围内单元格数据保存到指定文件中。成功返回真, 失败返回假。

参数<1>指定了欲保存到文件的单元格区域起始行号。行号从1开始。

参数<2>指定了欲保存到文件的单元格区域起始列号。列号从1开始。

参数<3>指定了欲保存到文件的单元格区域行数。如果本参数被省略, 默认值为1。

参数<4>指定了欲保存到文件的单元格区域列数。如果本参数被省略, 默认值为1。

参数<5>指定了保存数据的文件名。

例如:

数据源\_命令.单元格到文件 (1, 1, 3, 3, "c:\Bak.bak")

程序执行后将“数据源\_命令”中从(1, 1)起的3行3列的单元格区域中的数据保存到“C:\Bak.bak”

#### 62) “文件到单元格”命令

命令原型为:

<逻辑型> 对象. 文件到单元格 (整数型 行号, 整数型 列号, 文本型 单元格数据文件名, [逻辑型 自动扩展])

本命令用于将指定文件内的单元格数据写入到数据源中指定位置。成功返回真, 失败返回假。

参数<1>指定了读入的数据置入位置的起始行号。行号从1开始。

参数<2>指定了读入的数据置入位置的起始列号。列号从1开始。

参数<3>指定了保存数据的文件名。

参数<4>指定如果数据源内的数据行列数无法容纳即将读入的数据, 是否自动扩展。“真”为自动扩展, “假”为不自动扩展。如果本参数被省略, 默认值为真。

例如:

数据源\_命令.文件到单元格 (2, 2, "c:\Bak.bak", 真)

程序执行后将“C:\Bak.bak”的数据读入置入到“数据源\_命令”中的第二行第二列起的单元格中。

#### 63) “刷新”命令

命令原型为:

<无返回值> 对象. 刷新 ()

本命令用于通知所有基于本数据源的数据处理器, 使其自动从数据源重新获取并显示数据。







例如：

数据源\_命令.刷新 ()

数据源\_命令.保存更改 ()

程序执行后将“数据源\_命令”中的数据保存到数据提供者中并重新从数据提供者读取数据。

64) “保存更改”命令

命令原型为：

<无返回值> 对象.保存更改 ()

本命令用于通知所有基于本数据源的数据处理器，使其自动将处理或更改后的数据写回到数据源中。

例如：

数据源\_命令.保存更改 ()

程序执行后将“数据源\_命令”中的数据保存到数据提供者中。

65) “打印设置”命令

命令原型为：

<逻辑型> 对象.打印设置 ()

本命令用于调用对话框设置数据的打印配置信息。当操作者按确认按钮退出对话框时返回“真”，否则返回“假”。

例如：

数据源\_命令.打印设置 ()

程序执行后打开“数据源\_命令”的打印设置对话框。

66) “取打印设置”命令

命令原型为：

<打印设置信息> 对象.取打印设置 ()

本命令用于获取打印数据源数据时所将使用的设置信息。

例如：

变量名	类型	静态	数组	备注
打印信息	打印设置信息			

--- 如果真 (数据源1.打印设置 () = 真)

打印信息 = 数据源1.取打印设置 ()

打印信息.上边距 = 50

数据源1.置打印设置 (打印信息)

程序执行后打开打印设置对话框。如果用户在打印设置对话框中是点击“确定”按钮关闭的打印设置对话框，那么将所做的设置取出更改“上边距”为5毫米并设置更改后的



信息为当前打印设置。

67) “置打印设置” 命令

命令原型为：

<逻辑型> 对象. 置打印设置 (打印设置信息 打印设置信息)

本命令用于设置打印数据源数据时所将使用的设置信息。成功返回“真”，失败返回“假”。

参数<1>指定了欲设置的新信息。

例如：

变量名	类 型	静态	数组	备 注
打印信息	打印设置信息			

--- 如果真 (数据源1.打印设置 () = 真)  
打印信息 = 数据源1.取打印设置 ()  
打印信息.上边距 = 50  
数据源1.置打印设置 (打印信息)

程序执行后打开打印设置对话框。如果用户在打印设置对话框中是点击“确定”按钮关闭的打印设置对话框，那么将所做的设置取出更改“上边距”为5毫米并设置更改后的信息为当前打印设置。

68) “取打印页宽” 命令

命令原型为：

<整数型> 对象. 取打印页宽 ()

本命令用于获取打印数据源数据时所将使用打印纸的正文区域宽度，单位为0.1毫米。

**注解：**（1）此宽度不包含页边距；（2）“打印设置”命令执行后，如果操作者改变了纸张类型，此值将自动改变。

例如：

输出调试文本 (数据源\_命令.取打印页宽 ())  
输出调试文本 (数据源\_命令.取打印页高 ())

程序执行后在输出面板中显示现行“数据源\_命令”的打印页宽度和高度。

69) “取打印页高” 命令

命令原型为：

<整数型> 对象. 取打印页高 ()

本命令用于打印数据源数据时所将使用打印纸的正文区域高度，单位为0.1毫米。

**注解：**（1）此高度不包含页边距；（2）“打印设置”命令执行后，如果操作者改变了纸张类型，此值将自动改变。





例如：

输出调试文本(数据源\_命令.取打印页宽())

输出调试文本(数据源\_命令.取打印页高())

程序执行后在输出面板中显示现行“数据源\_命令”的打印页宽度和高度。

#### 70) “置只读方式”命令

命令原型为：

<无返回值> 对象. 置只读方式 (整数型 行号, 整数型 列号, [整数型 行数], [整数型 列数], 逻辑型 设置值)

本命令用于设置数据源中指定范围的单元格的是否只读。

**注解：** (1) 如果数据处理者不支持此特性，则本属性无效； (2) 如果数据提供者不支持此特性，本命令将被忽略。

参数<1>指定了欲设置只读方式的单元格范围起始行号。行号从1开始。

参数<2>指定了欲设置只读方式的单元格范围起始列号。列号从1开始。

参数<3>指定了欲设置只读方式的单元格范围行数。如果本参数被省略，默认值为1。

参数<4>指定了欲设置只读方式的单元格范围列数。如果本参数被省略，默认值为1。

参数<5>指定了范围内的单元格是否只读。如设置值为“真”，则数据源中指定范围单元格在表现时只能读取而不能写入。

例如：

数据源\_命令.置只读方式(1, 1, 1, 1, 真)

输出调试文本(数据源\_命令.取只读方式(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列的单元格为只读；在输出面板中显示“数据源\_命令”中第一行第一列的单元格是否为只读的。

#### 71) “取只读方式”命令

命令原型为：

<逻辑型> 对象. 取只读方式 (整数型 行号, 整数型 列号)

本命令用于获取数据源中指定单元格在表现时是否只读。返回“真”为只读。

参数<1>指定了欲测试的单元格所在行号。行号从1开始。

参数<2>指定了欲测试的单元格所在列号。列号从1开始。

例如：

数据源\_命令.置只读方式(1, 1, 1, 1, 真)

输出调试文本(数据源\_命令.取只读方式(1, 1))

程序执行后设置“数据源\_命令”中第一行第一列的单元格为只读；在输出面板中显示“数据源\_命令”中第一行第一列的单元格是否为只读的。





### 4.12.3 通用提供者

通用提供者用作为数据源提供对数据的通常存取操作。使用内存作为数据的存储仓库，全面支持所有数据操作接口。因此必要时可以将其他类型数据提供者内的数据转移到此类型中，以全面发挥数据源对数据的操纵能力。

#### 通用提供者属性

##### 1) “初始行数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定通用提供者的初始行数。

例如：

通用提供者\_属性.初始行数 = 10

通用提供者\_属性.初始列数 = 10

程序执行后将“通用提供者\_属性”的初始行数设为10行；将“通用提供者\_属性”的初始列数设为10列。

##### 2) “初始列数”属性

**属性类型：**整数型；**有效范围：**设计时、编程时；**编程时权限：**使用、读取、更改  
本属性用于指定通用提供者的初始列数。

例如：

通用提供者\_属性.初始行数 = 10

通用提供者\_属性.初始列数 = 10

程序执行后将“通用提供者\_属性”的初始行数设为10行；将“通用提供者\_属性”的初始列数设为10列。

## 4.13 核心库内置数据类型

在易语言中，很多支持库都拥有自己的内置数据类型。它们是与所在库功能相关的复合数据类型。库内置数据类型可以包含“成员”和“命令”。看上去有些类似自定义数据类型和类的对象（类和对象的概念本书后面的章节将详细讲解）。

下面介绍一下易语言核心支持库中的两个简单的内置数据类型——字体、打印设置信息。

**注解：**核心库中还有字段信息、对象、变体型等复杂内置数据类型。

关于“字段信息”参看本书“5.2 创建易数据库”章节。





关于“对象”和“变体型”参看本书“11.1 对象的应用”。

### 4.13.1 库内置数据类型的使用

库内置数据类型和基本数据类型、自定义数据类型的声明方法相同。  
 在前面的章节中我们已经接触过库内置数据类型，如按钮的“字体”属性。

变量名	类 型	静态	数组	备 注
变量1	字体			

```

变量1.加粗 = 真
变量1.字体名称 = “黑体”
按钮1.字体 = 变量1
    
```

库内置数据类型成员的使用与自定义数据类型相同。即 变量名.成员名。  
 库内置数据类型命令的使用与类的对象相似。即 变量名.命令名（[参数1]，[参数2] ...）。

### 4.13.2 字体

字体数据类型在本章中经常作为窗口组件的“字体”属性的值被使用，下面来详细了解一下。

成员

1) “角度”

**成员名称：**角度；**成员类型：**整数型；  
 本成员值对应字体样式的旋转角度。成员值以1/10度为单位，默认为0，既不旋转。

2) “加粗”

**成员名称：**加粗；**成员类型：**逻辑型；  
 本成员值指定了字体是否使用加粗效果。“真”为使用加粗，“假”为不使用。

3) “倾斜”

**成员名称：**倾斜；**成员类型：**逻辑型；  
 本成员指定了字体是否使用倾斜效果。“真”为使用倾斜，“假”为不使用。

4) “删除线”

**成员名称：**删除线；**成员类型：**逻辑型；  
 本成员指定了字体是否使用删除线效果。“真”为使用，“假”为不使用。

5) “下划线”

**成员名称：**下划线；**成员类型：**逻辑型；  
 本成员指定了字体是否使用下划线效果。“真”为使用，“假”为不使用。



## 6) “字体大小”

**成员名称：**字体大小；**成员类型：**整数型；

本成员指定了字体的大小。单位为点（1/72 英寸），初始值为“9”。

## 7) “字体名称”

**成员名称：**字体名称；**成员类型：**文本型；

本成员指定了字体所使用的字体名称。所指定的字体名称必须在系统中已安装，否则将有可能出错。成员初始值为“宋体”。

### 4.13.3 打印设置信息

本数据类型一般在打印相关操作时使用，本数据类型的变量保存了打印时所需要的设置信息。

#### 成员

## 1) “纸张类型”

**成员名称：**纸张类型；**成员类型：**整数型；

本成员设置了打印纸类型，可以为以下常量值之一：0（#默认纸）；1（#A3纸：297×420毫米）；2（#A4纸：210×297毫米）；3（#A5纸：148×210毫米）；4（#B4纸：250×354毫米）；5（#B5纸：182×257毫米）；6（#四开：215×275毫米）；7（#十六开：146×215毫米）；8（#三十二开：97×151毫米）；9（#信纸：216×279毫米）；10（#法律用纸：216×355毫米）；11（#行政用纸：184×266毫米）；12（#声明：140×216毫米）；13（#小报：279×432毫米）；14（#笔记：216×279毫米）；15（#账本：432×279毫米）；16（#对开纸：216×330毫米）。

除了以上基本类型纸张，还支持以下扩展类型，这些类型值和其所对应纸张宽度与高度（单位毫米）如下：

1001: 216×279、1002: 279×432、1003: 432×279、1004: 216×355、1005: 140×216、1006: 184×267、1007: 297×420、1008: 210×297、1010: 148×210、1011: 257×364、1012: 182×257、1013: 216×330、1014: 215×275、1015: 254×355、1016: 279×432、1017: 216×279、1018: 98×225、1019: 105×241、1020: 114×263、1021: 121×279、1022: 127×292、1023: 432×558、1024: 558×863、1025: 863×1117、1026: 110×220、1027: 162×229、1028: 324×458、1029: 229×324、1030: 114×162、1031: 114×229、1032: 250×353、1033: 176×250、1034: 176×125、1035: 110×230、1036: 98×190、1037: 92×165、1038: 378×279、1039: 216×305、1040: 216×330、1041: 250×353、1042: 100×148、1043: 228×279、1044: 254×279、1045: 381×279、1046: 220×220、1049: 241×305、1050: 241×381、1051: 305×457、1052: 235×322、1053: 216×279、1054: 210×297、1055: 241×305、1056: 227×356、1057: 305×487、







1058: 216×322、1059: 210×330、1060: 148×210、1061: 182×257、1062: 322×445、  
 1063: 174×235、1064: 201×276、1065: 420×594、1066: 297×420、1067: 322×445、  
 1068: 200×148、1069: 105×148、1070: 240×332、1071: 216×277、1072: 120×235、  
 1073: 90×205、1074: 279×216、1075: 420×297、1076: 297×210、1077: 210×148、  
 1078: 364×257、1079: 257×182、1080: 148×100、1081: 148×200、1082: 148×105、  
 1083: 332×240、1084: 277×216、1085: 235×120、1086: 205×90、1087: 128×182、  
 1088: 182×128、1089: 305×279、1090: 105×235、1091: 235×105、1092: 188×260、  
 1093: 130×184、1094: 140×203、1095: 102×165、1096: 102×176、1097: 125×176、  
 1098: 110×208、1099: 110×220、1100: 120×230、1101: 160×230、1102: 120×309、  
 1103: 229×324、1104: 324×458、1105: 260×188、1106: 184×130、1107: 203×140、  
 1108: 165×102、1109: 176×102、1110: 176×125、1111: 208×110、1112: 220×110、  
 1113: 230×120、1114: 230×160、1115: 309×120、1116: 324×229、1117: 458×324

**注解：**如果所选择纸张得不到打印机的支持，打印机将会自动选择最接近的纸张。

## 2) “纸张方向”

**成员名称：**纸张方向；**成员类型：**整数型；

本成员设置了打印纸的放置方向，可以为以下常量值之一：0（#纵向）；1（#横向）。

## 3) “左边距”

**成员名称：**左边距；**成员类型：**整数型；

本成员设置正文打印区到打印纸左边缘的距离，单位为0.1毫米，初始值为“230”。

## 4) “上边距”

**成员名称：**上边距；**成员类型：**整数型；

本成员设置正文打印区到打印纸上边缘的距离，单位为0.1毫米，初始值为“230”。

## 5) “右边距”

**成员名称：**右边距；**成员类型：**整数型；

本成员设置正文打印区到打印纸右边缘的距离，单位为0.1毫米，初始值为“230”。

## 6) “下边距”

**成员名称：**下边距；**成员类型：**整数型；

本成员设置正文打印区到打印纸下边缘的距离，单位为0.1毫米，初始值为“230”。

## 7) “页号位置”

**成员名称：**页号位置；**成员类型：**整数型；

本成员设置是否打印页号及页号的打印位置。

可以为以下常量值之一：0（无页号）；1（#上左页号）；2（#上中页号）；3（#上右页号）；4（#下左页号）；5（#下中页号）；6（#下右页号）。本成员初始值为“0”。



## 8) “打印份数”

**成员名称：**打印份数；**成员类型：**整数型；

本成员设置打印时共打印的份数。本成员初始值为“1”。

## 9) “首页打印页号”

**成员名称：**首页打印页号；**成员类型：**整数型；

本成员设置打印时首页的打印页号，其后的页号由本页号递增。本成员初始值为“1”。

## 10) “是否打印到文件”

**成员名称：**是否打印到文件；**成员类型：**逻辑型；

本成员设置是否将打印数据输出到指定文件。本成员初始值为“假”。

## 11) “打印文件名”

**成员名称：**打印文件名；**成员类型：**文本型；

如果设置为打印到文件，本成员指定数据输出到的文件名。

## 12) “自动填充”

**成员名称：**自动填充；**成员类型：**逻辑型；

本成员设置是否自动重复尾数据空行时填充尾页空白。注意如果“每页打印行数”成员值大于零，本成员无效。本成员初始值为“假”。

## 13) “自动添加表格线”

**成员名称：**自动添加表格线；**成员类型：**逻辑型；

本成员设置打印时是否自动为单元格添加边框线。“真”为添加，“假”为不添加。

## 14) “奇偶页方式”

**成员名称：**奇偶页方式；**成员类型：**整数型；

本成员可为以下常量值之一：0（#全部页）；1（#仅打印奇数页）；2（#仅打印偶数页）。本成员初始值为“0”。

## 15) “打印范围方式”

**成员名称：**打印范围方式；**成员类型：**整数型；

本成员可为以下常量值之一：0（#全部页）；1（#页范围）；2（#行范围）。本成员初始值为“0”。

## 16) “起始打印位置”

**成员名称：**起始打印位置；**成员类型：**整数型；

如果打印范围方式为“页范围”，本成员记录首页页号（页号从1开始）；如果打印范围方式为“行范围”，本成员记录首行行号；如果打印范围方式为“全部页”，本成员无效。本成员初始值为“1”。

## 17) “结束打印位置”

**成员名称：**结束打印位置；**成员类型：**整数型；





如果打印范围方式为“页范围”，本成员记录末页页号（页号从1开始）；如果打印范围方式为“行范围”，本成员记录末行行号；如果打印范围方式为“全部页”，本成员无效。如果本成员值为0，则将一直打印到所有数据行结束。本成员初始值为“0”。

#### 18) “每页打印行数”

**成员名称：**每页打印行数；**成员类型：**整数型；

本成员设置每页打印行数（不包含表头行），如果为0则表示根据页高自动判别。本成员初始值为“0”。

#### 19) “打印缩放比”

**成员名称：**打印缩放比；**成员类型：**整数型；

指定打印时所使用的缩放百分比，值为-1、-2或20~1000。其含义如下：

-1：#缩放到纸宽。即自动压缩或扩张到纸张宽度打印；

-2：#限定到纸宽。即：如果欲打印数据宽度超过纸张宽度，则自动压缩到纸张宽度，否则原样打印；

20~1000：直接指定打印缩放百分比；

如果指定为0，则默认为100。本成员初始值为“0”。

## 4.14 我的播放器（四）

在第四章中，学习了易语言核心支持库中的组件部分。接下来，将组件运用到MP3播放程序中，制作一个比较完整的MP3播放器。使“我的播放器”在控制播放的基础上增加播放进度的显示和对播放列表的操作。

程序中需要实现两个方面的功能：控制播放状态，包括播放MP3，暂停播放，继续播放，显示播放进度；操作播放列表，包括对播放列表的添加、删除、保存。添加组件完成这些功能，可以分六步来编写代码。

第一步：设计界面。

新建一个易语言程序，设计程序界面。加入按钮、标签、进度条、列表框等组件，分别设置各组件的属性；将“启动窗口”的标题改为“我的播放器（四）”。

按钮1标题设置为：播放，按钮1的名称设置为：播放按钮；用作播放MP3文件；

按钮2标题设置为：暂停，按钮2的名称设置为：暂停按钮；用作暂停播放MP3文件；

按钮3标题设置为：继续，按钮3的名称设置为：继续按钮；用作继续播放MP3文件；

按钮4标题设置为：添加，按钮4的名称设置为：添加按钮；用作向播放列表框中加入要播放的MP3文件名（全路径）；

按钮5标题设置为：删除，按钮5的名称设置为：删除按钮；用作删除播放列表框中的





MP3文件名;

按钮6标题设置为: 保存, 按钮6的名称设置为: 保存按钮; 用作保存播放列表框中的MP3文件名;

列表框1名称设置为: 播放列表框; 用作显示要播放的MP3文件名;

进度条1名称设置为: 播放进度条; 用作显示播放进度;

标签1标题设置为空; 标签1可视设置为假; 标签1名称设置为: 进度标签; 用作反馈播放进度。

调整各组件的大小和位置, 界面效果如图4-42所示。

界面设计完, 下面来编写代码

第二步: 添加删除播放列表。

双击“添加按钮”进入代码编辑区。

使用系统处理类命令中的“多文件对话框”命令打开目录选择MP3文件, 将选择的MP3文件名存放到“文本组变量”中, 并用“计次循环首”命令加入到“播放列表框”中。

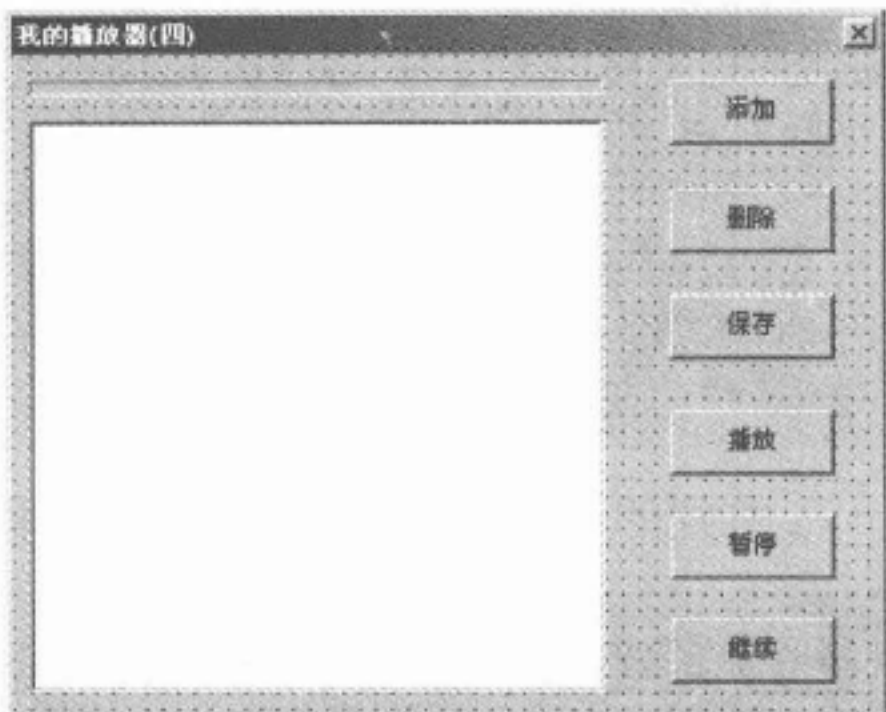


图4-42 界面效果

子程序名	返回值类型	公开	备注
_添加按钮_被单击			

变量名	类型	静态	数组	备注
文本组变量	文本型		0	

文本组变量 = 多文件对话框 (“添加mp3文件”, “mp3文件 (\*.mp3) | \*.mp3”, , , 真)

--- 如果真 (取数组成员数 (文本组变量) = 0)

返回 0

--- 计次循环首 (取数组成员数 (文本组变量), 计次变量)

播放列表框. 加入项目 (文本组变量 [计次变量], )

--- 计次循环尾 0

思考: 怎样判断重复添加的MP3文件?

“计次变量”在其他子程序中要用到, 所以要定义为程序集变量。

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
计次变量	整数型		

不想播放的MP3文件, 就从“播放列表框”中将其删除。

子程序名	返回值类型	公开	备注
_删除按钮_被单击			

播放列表框. 删除项目 (播放列表框. 现行选中项)



第三步：保存播放列表。

将“播放列表框”中的内容制作成播放列表保存起来，这样就不用每次使用MP3播放器都要添加MP3文件了。使用文件读写的方式来保存播放列表。首先“打开文件”，打开方式为“#重写”（写出数据到指定文件，如果该文件不存在则先创建一个新文件，如果已经存在就先清除其中的所有数据），返回文件的标识“文件号”；用“计次循环首”命令将“播放列表框”中的所有列表项目（即MP3文件名）取出，写到文件中；最后“关闭文件”。

子程序名	返回值类型	公开	备注
_保存按钮_被单击			

```
文件号 = 打开文件 (取运行目录 () + "\mp3.txt", #重写, )
--▶ 计次循环首 (播放列表框.取项目数 (), 计次变量)
    写文本行 (文件号, 播放列表框.取项目文本 (计次变量 - 1))
--- 计次循环尾 ()
关闭文件 (文件号)
```

用到的“文件号”也要定义为程序集变量，代码所下：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
计次变量	整型		
文件号	整型		

第四步：播放MP3，反馈播放进度。

对“播放列表框”中MP3文件的项目操作代码编写完后，来编写改变播放状态的代码。前面的例程中，使用的都是“播放MP3”命令，但是“播放MP3”命令与播放进度没有任何联系，而“同步播放MP3”命令可以通过标签的反馈事件取得播放进度，那么就使用“同步播放MP3”命令来播放MP3文件，并通过“进度标签”来反馈播放进度。在播放前要判断是否选中了“播放列表框”中的播放项目，没有选中，就设置选中第一项MP3文件。

子程序名	返回值类型	公开	备注
_播放按钮_被单击			

```
-- 如果真 (播放列表框.现行选中项 = -1)
    播放列表框.现行选中项 = 0
同步播放MP3 (播放列表框.取项目文本 (播放列表框.现行选中项), , 进度标签, )
```

思考：双击列表项怎样播放MP3文件？

在MP3文件播放过程中，会同步不断地给“进度标签”发送反馈事件通知，该事件的第一个参数提供现行播放百分比位置，为0~100。MP3文件播放完毕后，将固定收到一个播放百分比参数为100的事件通知。在“\_进度标签\_反馈事件”中，将参数一播放百分比赋值给“播放进度条”的位置属性，就可以看到“播放进度条”的进度位置发生了改变，“播放进度条”的位置进度就是该MP3文件的播放进度了。



那么，在MP3文件播放完毕，参数一的播放百分比到达100时，就需要自动播放下一首MP3文件。这时，就需要调整“播放列表框”的现行选中的播放项目，调整过后运行“\_播放按钮\_被单击”子程序继续播放。在调整中要判断：如果播放完毕的是“播放列表框”的第一项，下一个播放的就是第二项，即“现行选中项+1”，以后依此类推；如果播放完毕的是“播放列表框”的最后一项，下一个播放的就是第一项，即“现行选中项=0”，因为“列表框”的“现行选中项”属性是从0开始的。

子程序名	返回值类型	公开	备 注		
_进度标签_反馈事件	整型				
参数名	类 型	参考	可空	数组	备 注
参数一	整型				
参数二	整型				

播放进度条.位置 = 参数一

--- 如果真 (参数一 = 100)

--- 如果 (播放列表框.现行选中项 = 播放列表框.取项目数 () - 1)

--- 播放列表框.选择项目 (0, )

--- 播放列表框.选择项目 (播放列表框.现行选中项 + 1, )

--- 播放按钮\_被单击 ()

思考：怎样随意改变播放进度，从任意处播放MP3文件？

第五步：控制播放状态。

双击“暂停按钮”和“继续按钮”，实现暂停播放和继续播放，代码如下：

子程序名	返回值类型	公开	备注
_暂停按钮_被单击			

暂停播放MP3 ()

子程序名	返回值类型	公开	备注
_继续按钮_被单击			

继续播放MP3 ()

第六步：启动程序初始化。

操作播放项目的代码和操作播放状态的代码都已完成，但播放器的代码还有一个地方没写。前面编写了保存播放列表的代码，那么再次使用这个播放器时，就可以在程序启动时调入保存的播放列表。

在调入播放列表前，要先判断播放列表文件是否存在，如果不存在就先创建它。使用前面讲到的“打开文件”命令的“#重写”方式创建就可以，如果忘了可以看看前面“保存按钮”子程序的说明。对于已存在的播放列表文件直接读取它的内容就可以了。仍然是使用“打开文件”命令，但是这次用到的是“#读写”（从文件中读入数据或者写出数据到文件，如果该文件不存在则失败）方式，将读写位置“移到文件首”，使用“判断循环首”命令，判断读写位置“是否在文件尾”，不在文件尾，就根据读写位置“读入一行”文本，将该文本加入“播放列表框”中，每读入一行文本后，读写位置都要自动移动到该文本的后







面，然后继续判断进行循环；当判断的读写位置在文件尾时就结束循环，“关闭文件”。

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
文本变量	文本型			

```
--- 如果真 (文件是否存在 (取运行目录 () + "\mp3.txt") = 假)
    文件号 = 打开文件 (取运行目录 () + "\mp3.txt", #重写, )
    关闭文件 (文件号)
↓
文件号 = 打开文件 (取运行目录 () + "\mp3.txt", #读写, )
移到文件首 (文件号)
---▶ 判断循环首 (是否在文件尾 (文件号, 真) = 假)
    文本变量 = 读入一行 (文件号)
    播放列表框.加入项目 (文本变量, )
--- 判断循环尾 ()
关闭文件 (文件号)
```

思考：播放的MP3文件必须是全路径的，如果在“播放列表框”中只显示不带路径的MP3文件名，而又要保留MP3文件的路径用以播放，代码该怎么写？

### 4.15 小结

窗口是Windows窗口程序最显著的特征。窗口组件是拥有特殊样式的窗口。窗口是窗口组件和菜单的最基础容器。

易语言中的基本组件按照功能效果可以分为：按钮类、列表框类、系统类、图形类等，它们都是编写Windows窗口程序常用的窗口组件。

易语言中窗口拥有的属性、命令、事件被所有组件共有，所以称为共有属性、命令、事件；但具体到组件上是否有实际效果，应查看帮助文档或即时帮助。

### 4.16 习题

- 4-1 什么是组件？
- 4-2 易语言中窗口拥有的属性、命令、事件被称为\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_。
- 4-3 易语言中窗口组件的命令调用格式为\_\_\_\_\_属性值.命令名（[参数1]，[参数2]，...）。
- 4-4 更改窗口组件的途径有哪些？
- 4-5 易语言窗口组件在运行时是否都可以看见？



## 第三部分 易语言的数据库应用

在第三部分，将学习易语言对数据库的应用。包括易语言自有的数据库：易数据库的文件组成，创建方法，记录操作命令，数据库相关组件；另外还会介绍易语言对外部数据库的调用，SQL语言简介，ODBC连接数据库组件，ADO操作数据库组件。

通过这一部分的学习，掌握易语言访问各种数据库的能力，以便在程序中实现数据的存取和操作功能，从而完成更为复杂的编程需求。





## 第五章 易语言数据库的应用

### 本章目标

在本章结束时，我们能够：

- 了解什么是易数据库，易数据库的组成文件是什么
- 掌握如何使用菜单工具创建易数据库
- 掌握如何使用命令方式创建易数据库
- 掌握易数据库的“打开”，“关闭”的方法
- 掌握易数据库对记录的操作类命令
- 学会编写简单易数据库程序，熟练掌握添加、修改和删除记录的方法
- 了解易数据库相关组件和关联方法



## 5.1 了解易语言数据库

数据库（Database）是按照一定的数据结构来组织、存储和管理数据的仓库，它产生于距今五十年前，随着信息技术和市场的发展，特别是20世纪90年代以后，数据库的使用更加广泛。数据库有很多类型，从最简单的存储各种数据的表格到能够进行海量数据存储的大型数据库系统，在各个方面得到了广泛的应用，如：几乎所有人都在用的聊天工具QQ、MSN、邮件、企业中用到的OA、ERP等都是典型的数据库应用程序。易语言作为一款中国人自己研发的、真正本土化的编程软件，拥有完全自主知识产权和核心技术的数据库是必不可少的。易语言数据库，简称“易数据库”。易数据库是一个小型的数据库管理系统，它以一种简单的、类似表格的形式组织信息，并形成持久化存储。

易数据库的组成文件包括：

（1）扩展名为“.edb”的文件：易数据库主文件。

（2）扩展名为“.edt”的文件：易数据库辅助数据文件。仅在数据库中有备注型或者字节集型字段时才存在，文件名称除了后缀外与易数据库主文件相同，它必须与.edb文件放在同一目录中。

（3）扩展名为“.enx”的文件：易数据库索引文件。索引文件由用户根据需要自行创建，使用它可加快查找记录的速度。

下面通过一个易数据库程序的完整编写，逐步介绍易语言数据库的创建及具体使用方法。

## 5.2 创建易数据库

严格地说，数据库是“按照一定结构来组织、存储和管理数据的仓库”。在经济管理和日常工作中，常常需要把某些相关的数据放进这样的“仓库”，并根据管理的需要进行相应的处理。例如，人事部门常常要把单位员工的基本情况（姓名、性别、年龄、出生日期、工资、奖金等）存放在表中，这张表就可以看成是一个数据库。有了这个“数据仓库”就可以根据需要随时查询某员工的基本情况（见表5-1）。

表5-1 员工表

姓名	性别	年龄	出生日期	工资	奖金
李鹏	男	29	1980-5-14	1235	100
张林	男	27	1982-11-3	1300	220
王丽	女	32	1977-2-25	1500	98
刘飞飞	女	30	1979-12-20	2239	450





表5-1由表头（第一行）及数据组成。但具体用数据库来存储将如何操作呢？

易语言中建立数据库有两种方法：一种是使用易语言的编辑环境菜单“数据库”→“结构编辑器”创建，另一种是在程序中使用代码创建。

### 5.2.1 使用菜单创建易数据库

选择菜单“数据库”→“结构编辑器”，如图5-1所示。

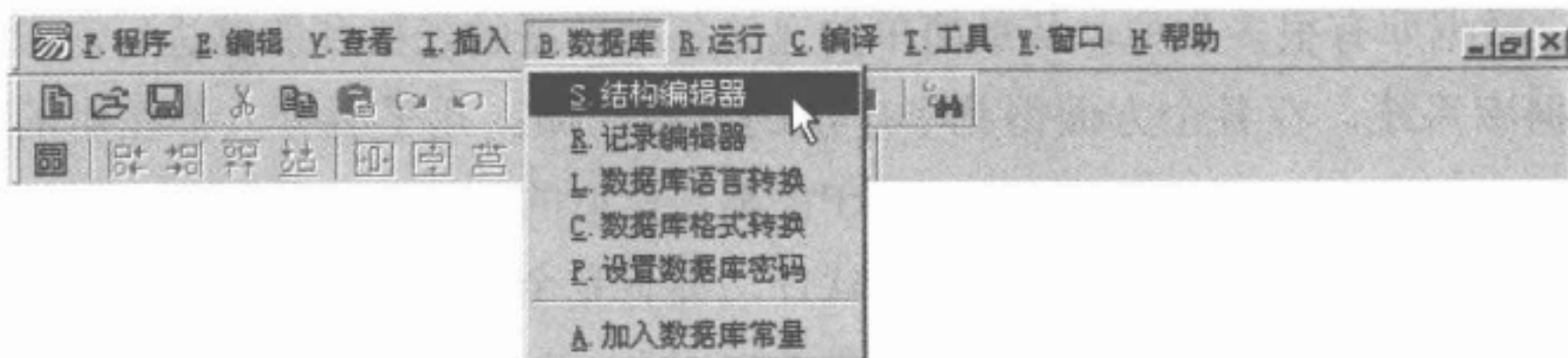


图5-1 菜单创建数据库

在下拉菜单中选择“结构编辑器”后会弹出数据库结构管理器界面，如图5-2所示。

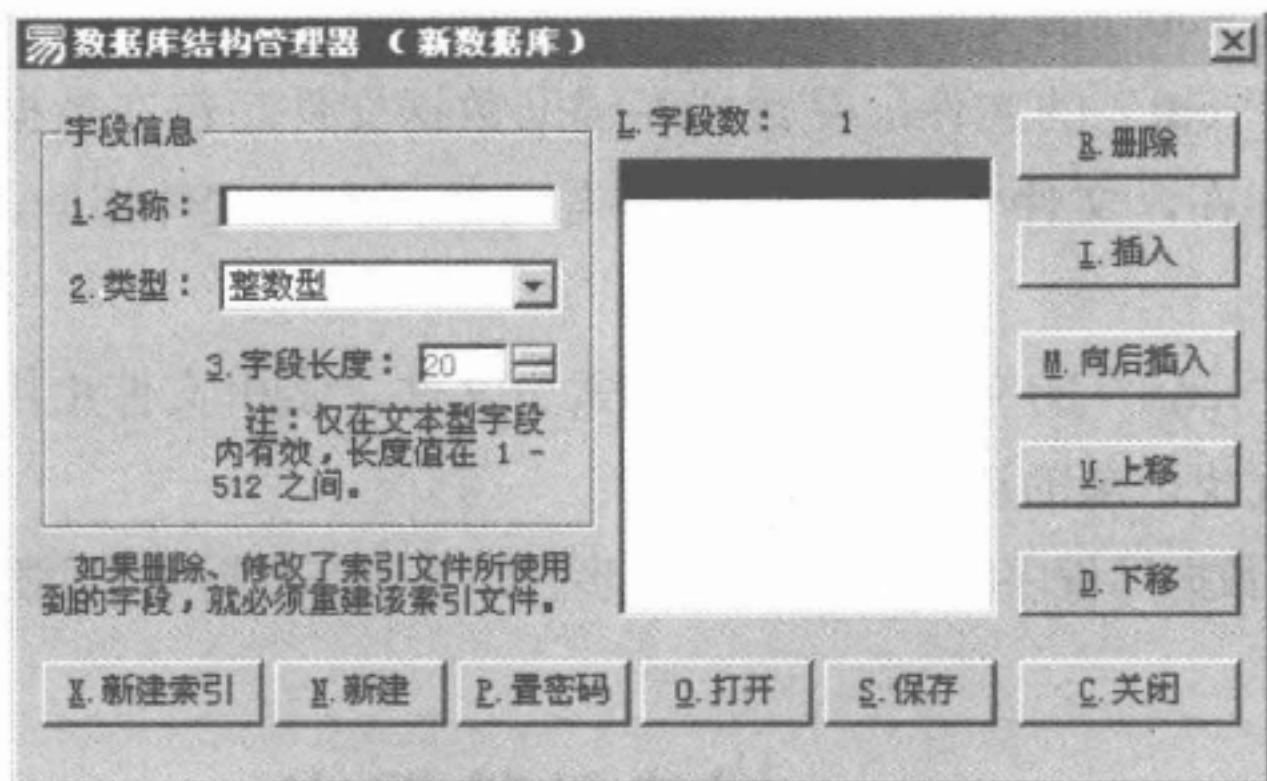


图5-2 数据库结构管理器

在这里首先要了解一下“字段信息”的概念，易数据库的每一个字段代表着数据表的一列（如表5-1中的每一列），字段在数据库中的属性称为“字段信息”（如表5-1表头）。“字段信息”是易语言核心支持库提供的库定义数据类型，它的结构见表5-2。

表5-2 字段信息的结构

成员	描 述
名称	文本型；名称文本的长度必须在 16 个字符以内。
类型	整数型；可以为以下常量值之一：1（#字节型）；2（#短整数型）；3（#整数型）；4（#长整数型）；5（#小数型）；6（#双精度小数型）；7（#逻辑型）；8（#日期时间型）；10（#文本型）；11（#字节集型）；12（#备注型）
最大文本长度	整数型；初始值为 20。本成员仅当字段类型为“文本型”时才有效，用作指定文本的最大可能长度，其值范围必须在 1 到 512 之间。如果字段类型不为“文本型”，本属性无效。当写入数据到数据库中的文本型字段时，超出的部分将被自动剪裁。

将表5-1的列对应设计成易数据库字段信息格式见表5-3。





表5-3 设计字段信息

字段名称	数据类型	文本长度	字段名称	数据类型	文本长度
姓名	文本型	8	出生日期	日期时间型	
性别	逻辑型		工资	小数型	
年龄	整数型		奖金	小数型	

在这里将“性别”字段设置为“逻辑型”；用“真”代表“男”，用“假”代表“女”。在易数据库中具有逻辑相反关系的字段可以设置为逻辑型。如“开”、“关”、“已婚”、“未婚”等。

依次建立这6个字段，并设置字段名称、类型、字段长度（仅文本型）三个属性，如图5-3所示。

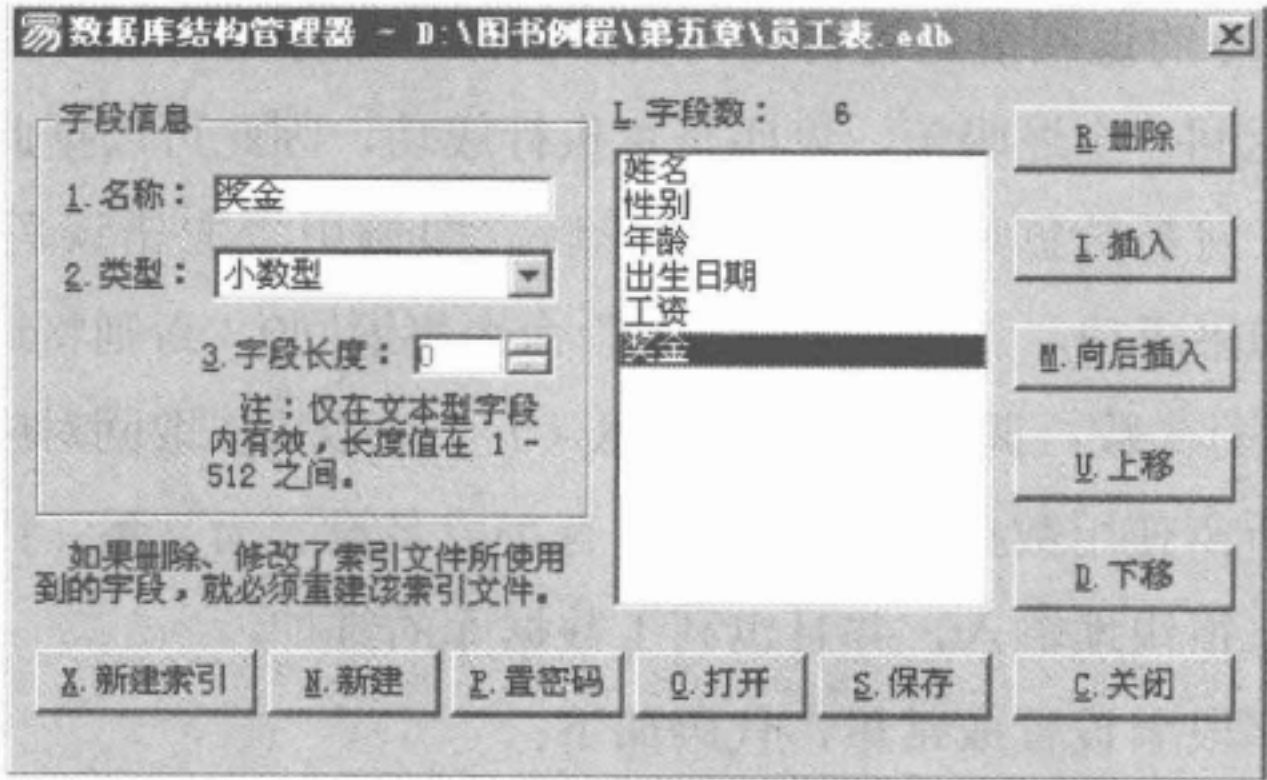


图5-3 定义数据库字段

点击保存，并为数据库命名，一个简单的人员信息的易数据库就建立了。

5.2.2 使用代码创建易数据库

另外，也可以在程序中使用代码动态地建立易数据库。

在代码中创建易数据库，需要使用“创建”命令。该命令是易语言核心支持库提供的易数据库类操作命令之一。易语言核心支持库提供的易数据库类操作命令有很多，包含了对易数据库的所有操作，如图5-4所示。

图5-4第一行显示的“数据库”指的是“易数据库”。本章如无特别说明，介绍的命令均是易数据库类操作命令。

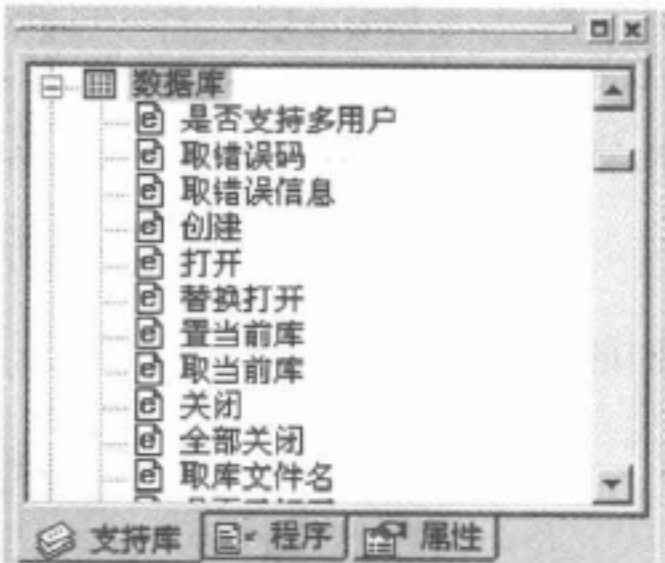


图5-4 易数据库类操作命令

创建（）命令  
命令原型为：

<逻辑型>创建（欲创建数据库的文件名，字段信息表）

“创建（）”命令创建指定的易数据库文件，如果该数据库文件存在，将被覆盖。成



功返回真，失败返回假。其参数描述见表5-4。

表5-4 参数描述

参数名	描 述
欲创建数据库的文件名	必需的；文本型
字段信息表	必需的；字段信息，参数数据只能提供数组数据。参数数组值中顺序指定本数据库中所有字段的信息

“创建（）”命令的第一个参数是欲创建数据库的文件名，如：“员工表.edb”；也可以加上欲创建的路径，如：“d:\mydata\员工表.edb”；如果只用数据库的文件名，那么此文件将被保存到当前易程序所在的目录中。

“创建（）”命令的返回值是逻辑型。实际上除少部分命令外，大多数的易数据库命令执行完毕后都会返回一个逻辑值，表明是否执行成功，因此可以据此判断执行的效果。

另外也可以在任何易数据库命令执行完毕后立即调用“取错误码（）”命令来查看其是否执行成功。如果成功，“取错误码（）”命令将返回0，否则将返回一个非0的错误值，如果发现命令执行失败，调用“取错误信息（）”命令可以取回对应的错误信息文本。

一般将程序中所需使用数据库的名称及所有字段名设置为常量，因为常量可以使用系统内置的输入提示来很快地输入，并且也利于数据库的维护。

将所需添加的字段名设置成常量，代码如下：

常量名称	常量值	公开	备 注
员工表	“员工表.edb”		
姓名	“姓名”		
性别	“性别”		
年龄	“年龄”		
出生日期	“出生日期”		
工资	“工资”		
奖金	“奖金”		

对于已经存在的易数据库，可以使用菜单“数据库”→“加入数据库常量”，一次性地将易数据库中的字段名称加入常量表，如图5-5所示。

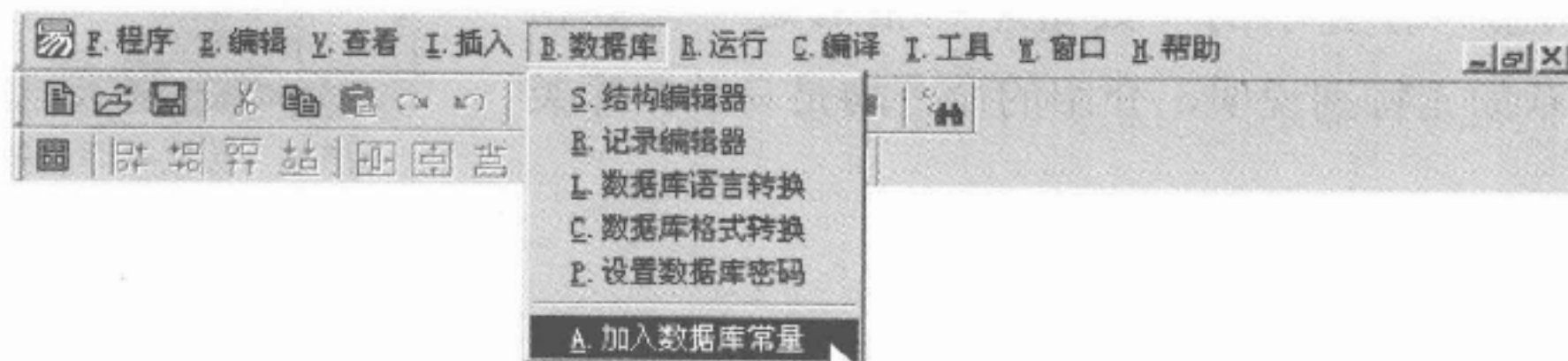


图5-5 加入数据库常量

选择需要添加的易数据库，方法如图5-6所示。



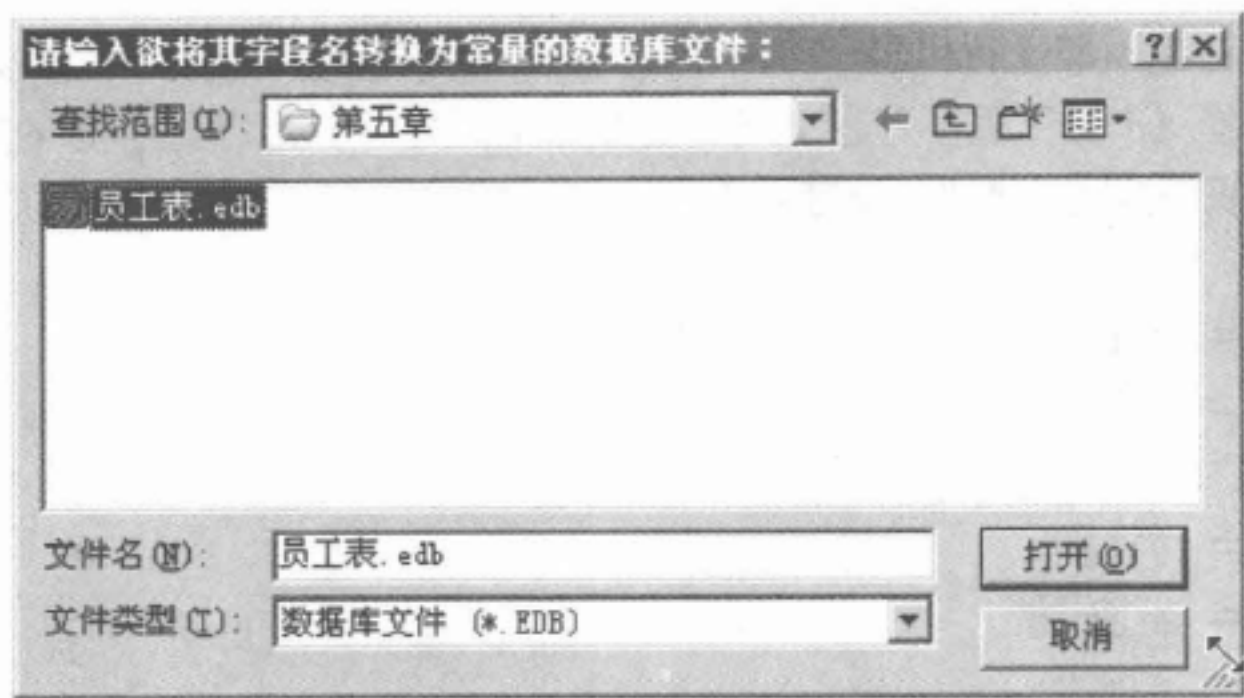


图5-6 选择欲加入常量的数据库

加入后的效果如下：

常量名称	常量值	公开	备注
数据库1	“员工表”		
姓名	“姓名”		
性别	“性别”		
年龄	“年龄”		
出生日期	“出生日期”		
工资	“工资”		
奖金	“奖金”		

这里将数据库名称更改为“员工表”，将数据库常量值更改为“员工表.edb”。

**注意：**在数据库常量值里要加上扩展名“.edb”，否则打开数据库时会寻找不到。

修改后如下：

常量名称	常量值	公开	备注
员工表	“员工表.edb”		
姓名	“姓名”		
性别	“性别”		
年龄	“年龄”		
出生日期	“出生日期”		
工资	“工资”		
奖金	“奖金”		

使用“创建（）”命令来创建易数据库：新建一个子程序“子程序\_建数据库”，在子程序中输入如下代码：

子程序名	返回值类型	公开	备注
子程序_建数据库			

变量名	类型	静态	数组	备注
字段表	字段信息		6	





```

--- 如果真 (文件是否存在 (取运行目录 () + “\” + #员工表) = 假)
--- 如果真 (信息框 (“员工表不存在, 是否创建数据库?”, #询问图标 + #是否钮, ) = #否钮)
    信息框 (“系统将会退出?”, #错误图标, )
    结束 ()
--- 字段表 [1].名称 = #姓名
    字段表 [1].类型 = #文本型
    字段表 [1].最大文本长度 = 8
    字段表 [2].名称 = #性别
    字段表 [2].类型 = #逻辑型
    字段表 [3].名称 = #年龄
    字段表 [3].类型 = #整数型
    字段表 [4].名称 = #出生日期
    字段表 [4].类型 = #日期时间型
    字段表 [5].名称 = #工资
    字段表 [5].类型 = #小数型
    字段表 [6].名称 = #奖金
    字段表 [6].类型 = #小数型
--- 如果真 (创建 (取运行目录 () + “\” + #员工表, 字段表) = 假)
    信息框 (“创建员工表数据库失败?”, #错误图标, “错误”)
    结束 ()
--- 信息框 (#左引号 + #员工表 + #右引号 + “数据库创建成功?”, #信息图标, )

```

在这段程序中首先判断“员工表”数据库是否存在, 如果存在则不执行, 否则创建“员工表”。

这样一个易数据库就创建了。

### 5.3 为易数据库添加记录

易数据库创建之后, 可以为其添加记录。

添加记录有菜单添加和代码添加, 首先介绍如何使用菜单添加。

选择菜单“数据库”→“记录编辑器”, 如图5-7所示。

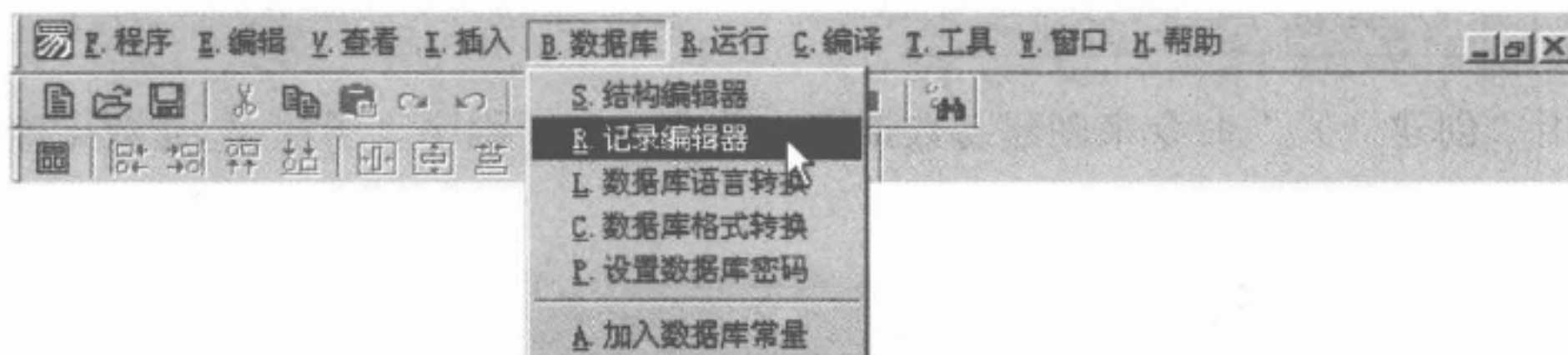


图5-7 选择记录编辑器

选择刚才创建的“员工表”数据库, 如图5-8所示。







图5-8 选择欲编辑其记录的数据库

打开之后，通过菜单可以进行数据的添加、删除等操作，如图5-9所示。

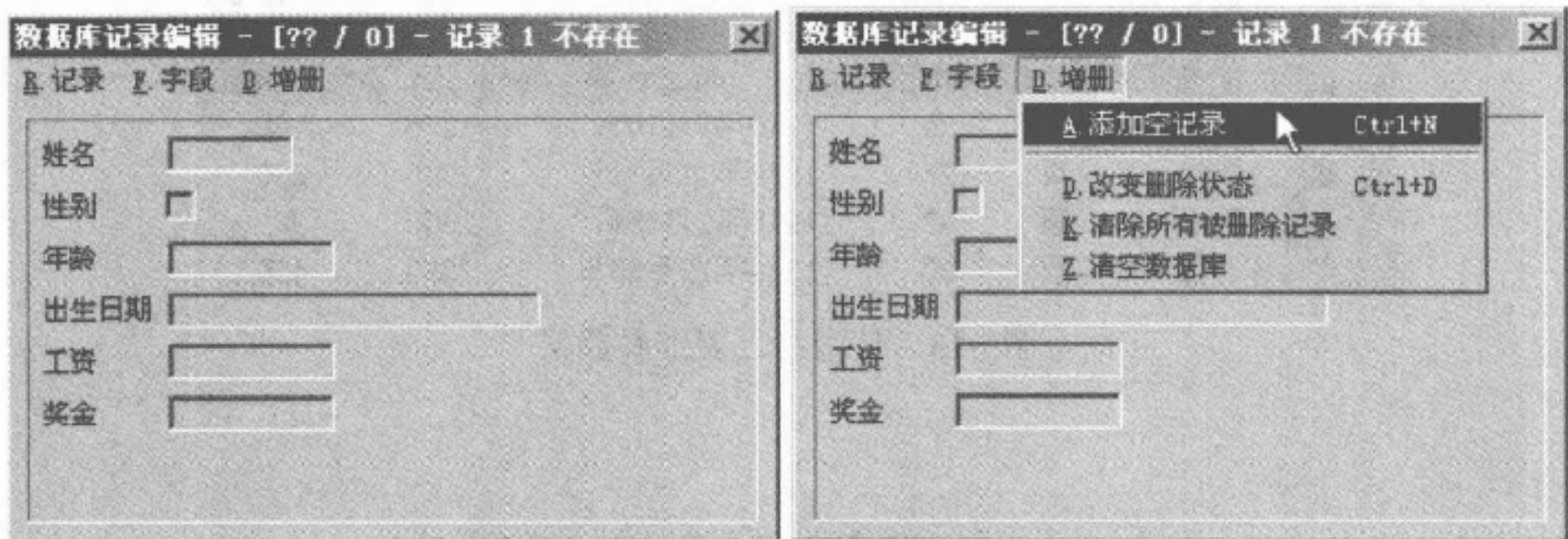


图5-9 添加空记录

添加一条空记录后，就可以输入该记录各字段的数据了，如图5-10所示。

通过添加空记录，设置该记录各字段的数据，依次将表5-1所示的四条记录添加到数据库中去。

要查看全部添加好的记录，可以使用易语言菜单“工具”中提供的“报表编辑器”。

首先，选择菜单“工具”→“报表编辑器”，如图5-11所示。

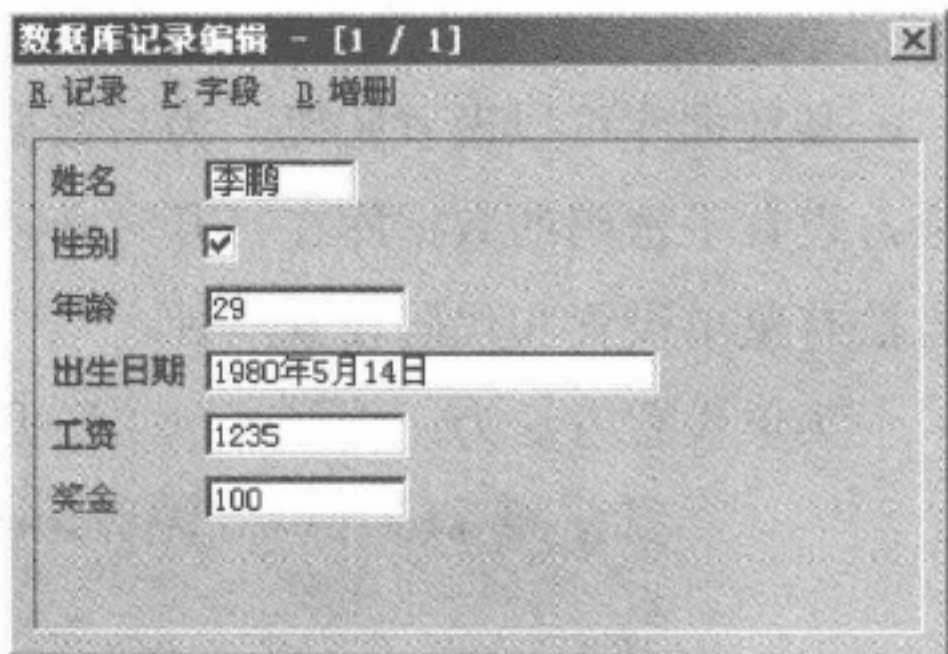


图5-10 编辑记录

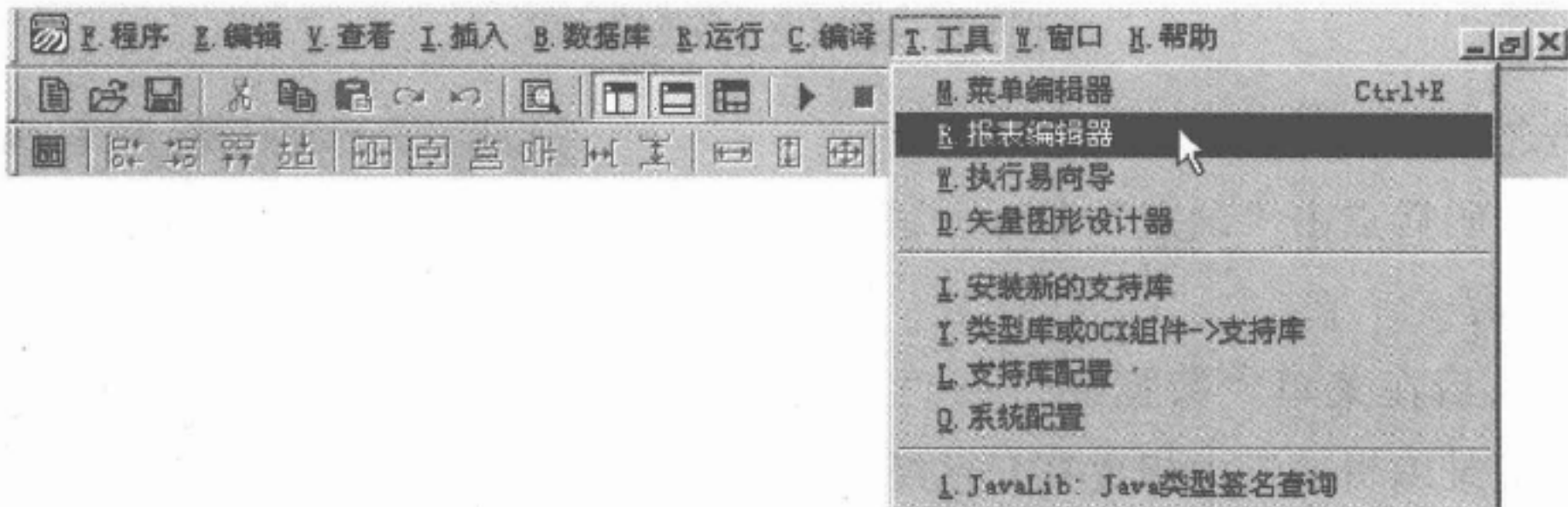


图5-11 选择报表编辑器



选择“报表编辑器”将显示出“易之表”工具，在其菜单中选择“文件”→“打开数据库”菜单，如图5-12所示。

选择“员工表”数据库，数据库将会被打开并显示全部记录，效果如图5-13所示。

可以在这个界面中进行修改记录的操作，退出时数据就自动保存到“员工表”数据库中了。

在程序中如何使用代码编写实现记录的添加，将在后面的“添加数据”功能的实现部分介绍。

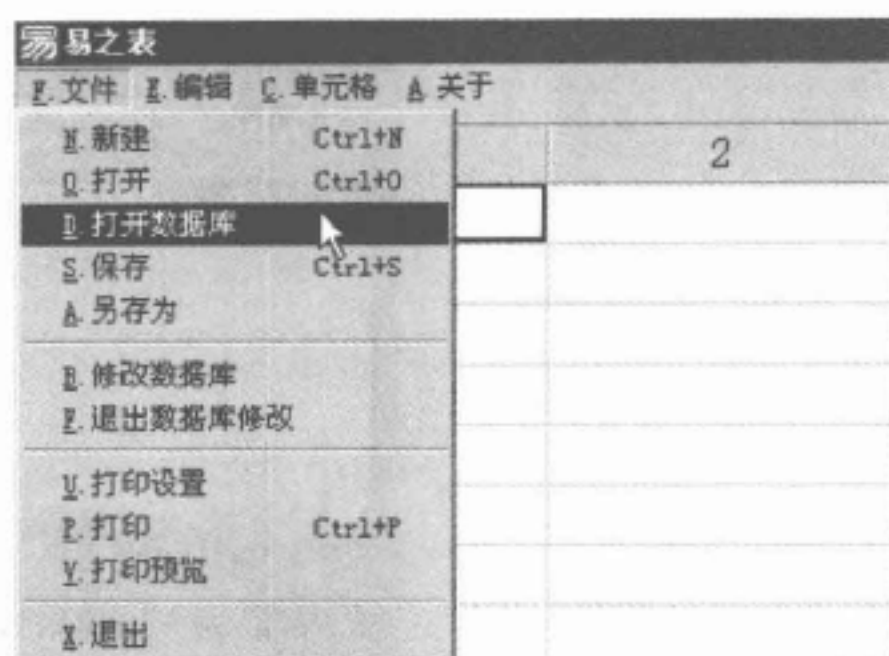


图5-12 使用易之表打开数据库

易之表						
文件 编辑 单元格 关于						
	姓名	性别	年龄	出生日期	工资	奖金
1	李鹏	真	29	1980年5月14日	1235	100
2	张林	真	27	1982年11月3日	1300	220
3	王丽	假	32	1977年2月25日	1500	98
4	刘飞飞	假	30	1979年12月20日	2239	450

图5-13 使用易之表查看数据

## 5.4 易数据库密码的设置

易数据库可以设置密码，为易数据库设置密码可以提高数据库的安全性，避免信息外泄，设置了密码的数据库在打开时必须提供正确的密码才可打开。易数据库密码的设置方法有菜单设置和代码设置两种。

菜单设置方法为：选择菜单“数据库”→“设置数据库密码”，如图5-14所示。

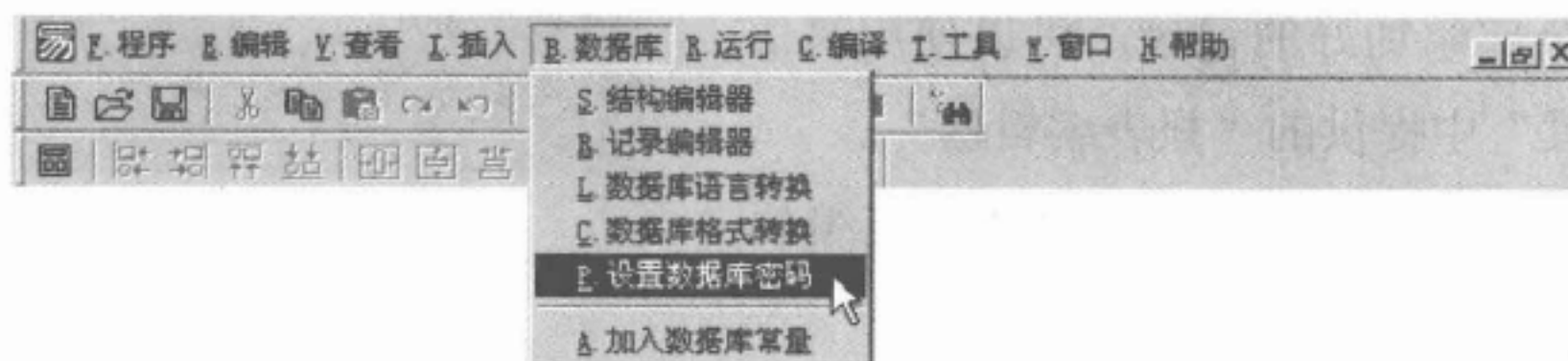


图5-14 选择设置数据库密码

点击之后显示界面如图5-15所示。

输入密码后点击“确定”即可，这样该数据库就有密码了。

如果以后在菜单“数据库”→“结构编辑器”或“记录编辑器”中想要打开已经存在的数据库时，就会提示需要输入相关密码，只有输入

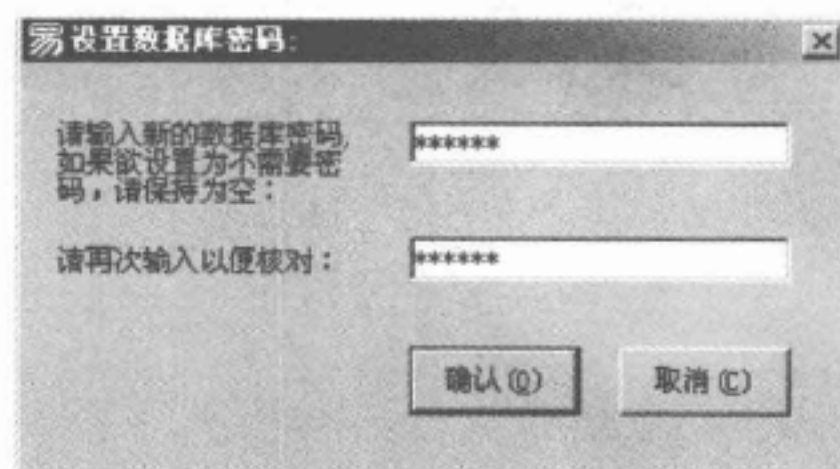


图5-15 设置数据库密码



了正确的密码才可以打开。

使用代码也同样可以设置数据库密码，相应的对于有密码的数据库，在程序中打开时也要提供相关密码。

使用代码设置数据库密码需要使用置数据库密码（）命令。

置数据库密码（）命令

命令原型为：

<逻辑型> 置数据库密码（[文本型 新密码文本]）

“置数据库密码（）”命令设置或修改当前数据库的访问密码。

**注意：**数据库文件必须可写。成功返回真，失败返回假。其参数描述见表5-5。

表5-5 参数描述

参数名	描 述
新密码文本	文本型，可以被省略 建议密码长度最少为6个字符，并且最好混合使用字母、数字、符号，以保证密码本身的安全性 注意不能为“?”（即单个问号），此密码文本被保留用作其他用途 如果为空文本，表示无密码 如果本参数被省略，则默认为空文本

在设置数据库密码之前可以使用“是否已加密（）”命令来判断数据库是否已经被加密。

是否已加密（）命令

命令原型为：

<逻辑型> 是否已加密（文本型 数据库文件名）

“是否已加密（）”命令判断是否已经对数据库进行了加密。如果已经被加密，返回真，否则返回假。其参数描述见表5-6。

表5-6 参数描述

参数名	描 述
数据库文件名	必需的：文本型

例如：

```
如果 (是否已加密 (取运行目录 () + "\数据库1.edb") = 假)
    如果真 (打开 (取运行目录 () + "\数据库1.edb", "数据库1", , , , ) = 假)
        信息框 ("数据库打开失败!", #错误图标, )
        结束 ()
    如果真 (置数据库密码 ("123456") = 假)
        信息框 ("设置数据库密码失败!", #错误图标, )
        结束 ()
```





这一段代码专门用来对“数据库1.edb”进行加密。首先判断“数据库1.edb”是否已经有密码，使用的是“是否已加密（）”命令。如果已经有密码则什么也不做，如果没有密码，使用“打开（）”命令打开数据库，然后使用“置数据库密码（）”命令为“数据库1.edb”设置密码。

在设置易数据库密码时数据库一定要处于“打开”状态，否则会失败，“打开（）”命令将在后面介绍。

## 5.5 易数据库相关组件

### 5.5.1 易数据库相关组件分类

为了方便地将数据库中的数据显示到程序界面，易语言提出了“数据应用框架”的概念。

“数据应用框架”最大的特点就是把数据、数据操作、数据显示分为三个不同的层次。每个层次由各自的组件完成相对独立的工作，至于各层次之间的联系，则由易语言在内部实现。这三个层次由低到高分别是：数据提供者、数据源、数据处理者（即显示数据的组件）。其中，数据提供者用于存储、提供数据；数据源用于操作数据库；数据处理者用于显示数据。

在易语言中，可充当数据提供者的组件有“通用提供者”、“数据库提供者”、“外部数据提供者”等（“外部数据提供者”组件将在下一章中介绍）；可充当数据源的只有“数据源”组件；可充当数据处理者的组件有“表格”、“编辑框”、“标签”、“列表框”组件等，见表5-7。

表5-7 易数据库组件分类

层次	类 别
数据处理者	表格、编辑框、标签、图片框、组合框、列表框、选择框、日期框
数据源	数据源组件
数据提供者	通用提供者、数据库提供者、外部数据提供者

### 5.5.2 易数据库相关组件介绍

#### 1) 通用提供者简介

“通用提供者”可作为数据源提供对数据的存取操作。使用内存作为数据的存储仓库，全面支持所有数据操作接口。因此必要时可以将其他类型数据提供者的数据转移到此类型中，以全面发挥数据源对数据的操纵能力。

“通用提供者”的重要属性有“初始行数”和“初始列数”，用于指定初始数据的行数和列数。





2) 数据库提供者简介

“数据库提供者”可作为数据源提供对数据库的存取操作。它使用数据库作为数据的存储仓库，不支持以下数据操作接口：1（置行高）；2（置类型）；3（置文本色）；4（置背景色）；5（置字体名）；6（置字体尺寸）；7（置字体属性）；8（置边距）；9（置文本输入格式）；10（置对齐方式）；11（置密码方式）；12（合并）；13（分解）；14（加线条）；15（删线条）；16（初始尺寸时同时改变列数）；17（在中间插入行）；18（插入列）；19（删除列）。如果想对数据进行以上操作，应该将数据通过数据源“导入”到通用提供者中。

将数据库提供者中的数据“导入”到通用数据提供者中，方法很简单，只需用数据源组件的“添加（）”命令就可以实现：

数据源1.添加(数据库提供者1,,)

具体例程见随书例程：“\图书例程\第五章\导入数据库提供者数据.e”文件。

“数据库提供者”有“数据库文件名”属性，用于指定与数据库提供者绑定的数据库文件名。

“字节集字段处理”属性，用于指定对字节集类型字段的处理方式。

“数据库密码”属性，用于提供访问数据库时所需要的密码。

“通用提供者”和“数据库提供者”的区别：相对而言，“通用提供者”功能强大，可以对数据进行各种操作，但没有直接的数据来源，“数据库提供者”功能较少，只能完成数据的基本操作，但可以直接连接到数据库。

3) 数据源简介

“数据源”一般用于配合各种数据提供者窗口组件提供数据。“数据源”组件中内置了一系列对数据库的操作，可以简化窗口组件与数据库的关联。

“数据源”的另一个主要功能是和表格相连，进行各种报表操作。

“数据源”提供对各种来源数据的统一操作接口。“数据源”组件各个按钮（如图5-16）的含义分别是：到首记录、到上一记录、到下一记录、到尾记录、添加新记录、删除当前记录。



图5-16 数据源组件

“数据源”的“数据提供者”属性，指定本数据源所基于的“数据提供者”组件名。在属性面板中设置该属性时，如果此时窗体上已经放置了某个（或多个）数据提供者组件，则会以下拉列表的形式列出，选择其中之一即可。

如果要在程序中用代码的方式为该属性赋值，需以文本形式指明“数据提供者”组件名称。

例如：

数据源1.数据提供者 = “通用提供者1”

“数据源”组件的命令非常多（如图5-17），将在后面的“易数据库操作例程”中介绍常用部分。

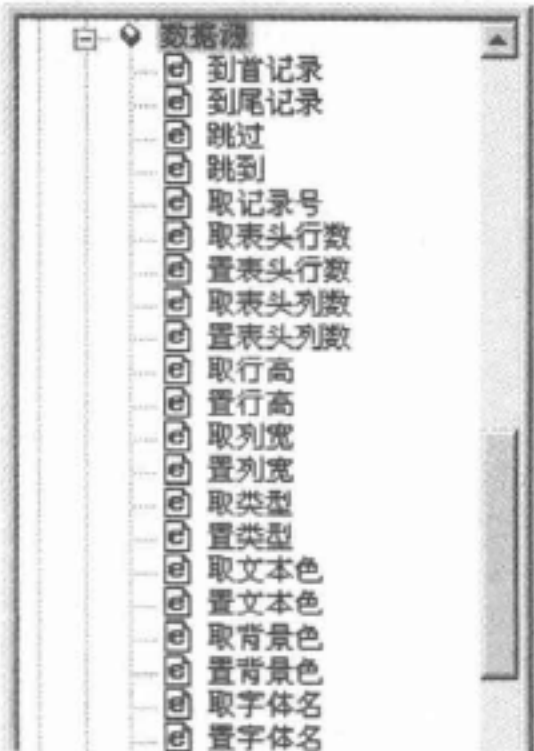


图5-17 数据源的命令



表格、数据源、通用提供者、数据库提供者的属性、命令、事件等更详细的讲解参看“第二部分 易语言的命令与组件”中“4.11 显示类组件”、“4.12 数据类组件”相关内容。

### 5.5.3 易数据库组件关联方法

数据提供者、数据源、数据处理者三者之间必须事先设置关联，才能共同完成对数据的处理。

设置方法：首先一次性添加相应的组件，数据处理者、数据源、数据提供者这三种类型的组件必须同时存在。然后将数据源的“数据提供者”属性设置为某个数据提供者组件；最后将数据处理者组件的“数据源”属性设置为某个数据源组件。

下面举一个使用“数据库提供者”的例子。

新建一个易程序，添加“数据库提供者”、“数据源”、“表格”组件各一个。

选择“数据库提供者1”的“数据库文件名”属性。此处选择前面用到的“员工表”，如图5-18所示。

设置数据源的“数据提供者”属性，为刚设置完的“数据库提供者1”，如图5-19所示。

设置表格的“数据源”属性，为上面刚设置的“数据源1”，如图5-20所示。

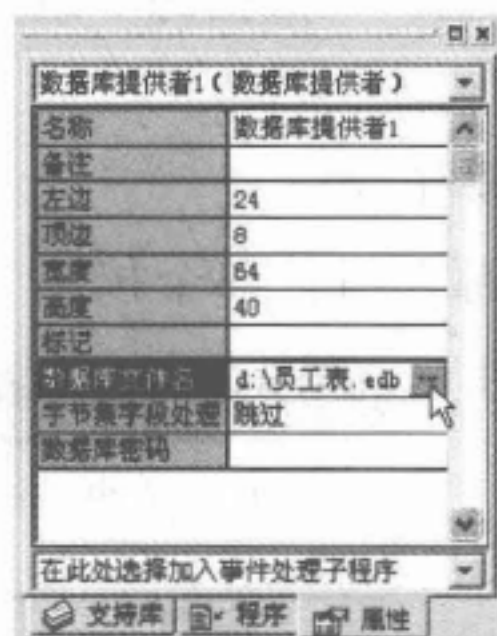


图5-18 设置数据库提供者



图5-19 设置数据源



图5-20 设置表格

表格的“数据源”属性设置完之后，表格内容显示如图5-21。

	姓名	性别	年龄	出生日期	工资	奖金
1	李小鹏	真	29	1980年5月14日	1235	100
2	张林	真	27	1982年11月3日	1300	220
3	王丽	假	32	1977年2月25日	1500	98
4	刘飞飞	假	30	1979年12月20日	2239	450

图5-21 表格显示数据库内容

正是数据库“员工表.edb”的内容。具体例程见随书例程：“\图书例程\第五章\数据库提供者.e”文件。



## 5.6 程序界面设计与组件关联

### 5.6.1 易数据库程序的界面设计

现在开始使用前面学到的知识，编写一个易数据库程序。

具体例程见随书例程：“\图书例程\第五章\易数据库操作例程.e”。

第一步：画出程序的界面：先添加一个“通用提供者”组件，将“初始行数”和“初始列数”分别定义为1和6。添加一个“数据源”组件，将其“数据提供者”属性设置为刚添加的“通用提供者1”，添加一个“表格”组件，用来显示记录，将“表格”的“数据源”属性设置为刚添加的“数据源1”组件。

在对数据的操作方面将使用易语言核心支持库提供的易数据库类操作命令，在对数据的显示方面将使用“数据源”组件的方式。易数据库和数据源组件的相关命令将在后面的实际代码编写中详细介绍。

由于使用了表格组件显示数据，所以要使用数据源和通用提供者提供数据。

第二步：添加7个按钮，按钮的标题设置为：“创建易数据库”、“打开易数据库”、“显示全部数据”、“添加数据”、“修改数据”、“删除数据”、“打印数据”，分别代表其字面对应的7个功能。

在“添加数据”被单击后，需要弹出一个界面，用户在该界面输入数据，保存并返回。在“修改数据”被单击后，也要弹出一个界面，用户在该界面修改数据，保存并返回。（这两个界面不做详细介绍，读者可以看提供的例程。）

第三步：添加4个按钮，按钮的标题设置为：“到首记录”、“上一条”、“下一条”、“到尾记录”，分别代表使用记录指针对某一条记录的显示。

第四步：添加一个标签、两个按钮，分别显示“欲寻找姓名”、“查找”、“继续查找”，一个编辑框用来输入欲查找的姓名。这几个组件完成查找数据并显示的功能。

这样程序界面就设计完成，如图5-22所示。

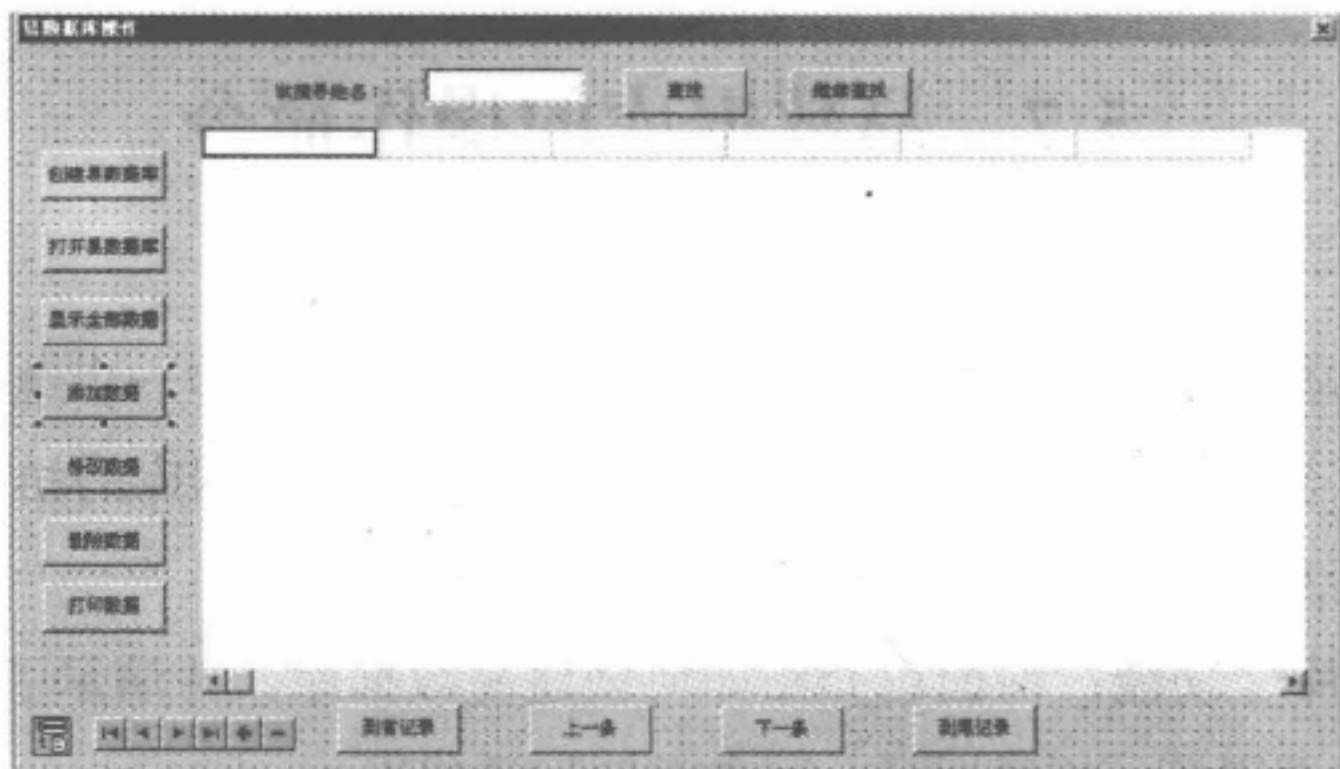


图5-22 易数据库操作例程界面



**说明：**本程序只是例程，用来帮助读者学习易数据库的知识，所以设计得尽量简单并不完善。读者在运行该程序的时候，要遵循以下顺序：

- (1) 数据库不存在的情况下要先点击按钮“创建易数据库”来创建数据库。
- (2) 数据库存在的情况下要先打开易数据库，才可进行数据库的操作。
- (3) “添加数据”、“修改数据”、“删除数据”、“查找”的操作之前要点击“显示全部数据”按钮在全部数据显示的情况状态下进行。

### 5.6.2 易数据库程序的组件关联

欲将数据库相关组件关联，首先设置“通用提供者1”的“初始行数”和“初始列数”属性分别为1和6，表示表格要显示的数据是6列1行，如图5-23所示。

设置“数据源”的“数据提供者”属性为刚设置完的“通用提供者1”，如图5-24所示。

设置“表格”的“数据源”属性为前面刚设置的“数据源1”，如图5-25所示。



图5-23 设置通用提供者



图5-24 设置数据源

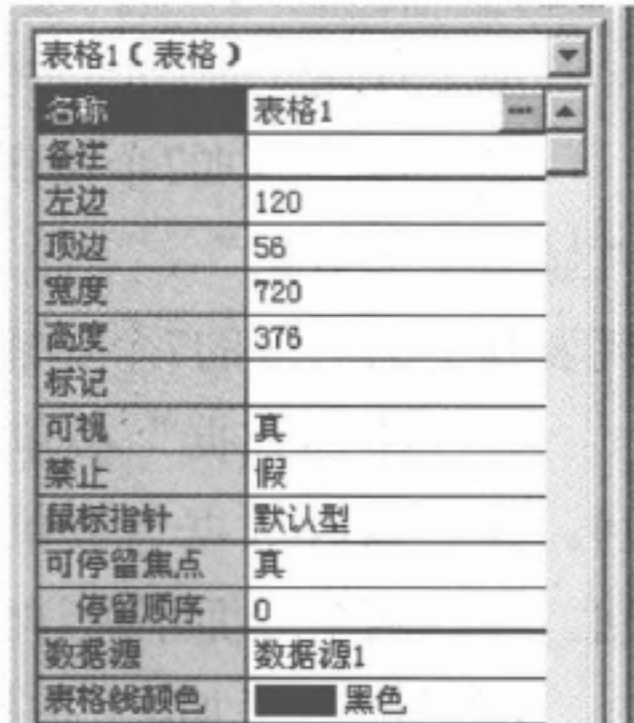


图5-25 设置表格

这样数据库相关组件就设置好了。在程序中就可以很方便地使用“数据源”将数据显示到“表格”中。

## 5.7 易数据库的操作命令

下面通过“易数据库操作例程”各部分功能的实现，来一步一步地讲解易语言的数据库相关知识，读者学习完本例程的编写之后，也就掌握了易数据库的知识。

将从程序界面左侧第一个按钮“创建易数据库”讲起。

“创建易数据库”功能：前面已介绍如何创建易数据库，并演示了“子程序\_建数据库”子程序，直接调用该子程序即可，可参考例程源代码。



### 5.7.1 数据库的打开与关闭

#### 1) 打开易数据库

易数据库在使用前需要先打开，不打开则无法使用，已经打开的数据库不能够重复打开。

欲打开指定数据库，可使用“打开（）”命令。

命令原型为：

<逻辑型> 打开（文本型 数据库文件名，[文本型 在程序中使用的别名]，[逻辑型 是否只读]，[整数型 共享方式]，[文本型 保留参数1]，[文本型 数据库密码]，[文本型数组/非数组 索引文件表]，...）

“打开（）”命令用于打开指定的数据库文件。成功返回真，并自动将“当前数据库”状态指向此数据库，失败返回假。其参数描述见表5-8。“当前数据库”状态的含义在下文会讲到。

表5-8 参数描述

参数名	描 述
数据库文件名	必需的；文本型
在程序中使用的别名	文本型，可以被省略。如果本参数被省略，默认为没有别名
是否只读	逻辑型，可以被省略。参数值说明是否仅对数据库进行读操作，如果省略本参数，默认为假
共享方式	整数型，可以被省略 参数值指定在多用户环境中限制其他用户操作此数据库的方式
保留参数1	文本型，可以被省略
数据库密码	文本型，可以被省略。本参数提供访问数据库时所需要的密码
索引文件表	文本型，可以被省略，提供参数数据时可以同时提供数组或非数组数据

“别名”的含义：“别名”为在后面的程序中引用本数据库时可使用的另一个名称。欲引用一个已经被打开的数据库可以使用该数据库本身的名称（数据库名称为数据库文件名的无路径和后缀部分。譬如 c:\my documents\yg.edb，其数据库名称为 yg），也可以使用在此处所指定的“别名”。“别名”主要用作避开重复的数据库名称或者简化对长数据库名的引用。数据库名称与“别名”皆不区分大小写，在查找数据库时，系统将优先查找“别名”。在“打开易数据库”按钮的单击事件中输入代码如下：

子程序名	返回值类型	公开	备注
按钮_打开_被单击			

如果 (打开 (#员工表, “员工表”, , , , ) = 假)

信息框 (“员工数据库打开失败?”, #错误图标, “错误”)

结束 ()

信息框 (“员工数据库打开成功?”, #信息图标, “成功”)

在这里设置数据库“员工表.edb”的别名为“员工表”。





运行程序，在单击“打开数据库”按钮后，程序会显示打开成功的字样，如图5-26所示。

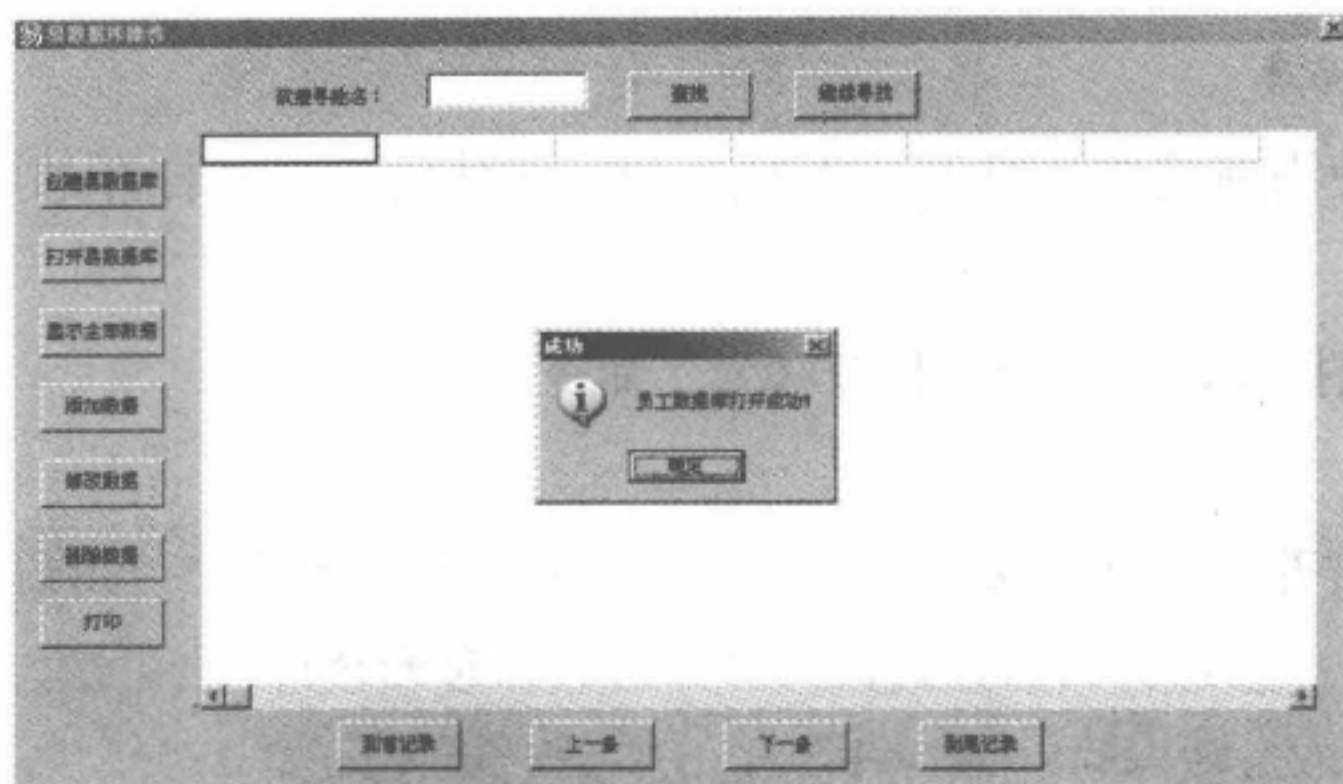


图5-26 打开数据库成功

如果重复打开或数据库文件不存在，会提示打开数据库失败，如图5-27所示。



图5-27 打开数据库失败

在一个易语言程序中，可以同时打开多个数据库，即多次使用“打开（）”命令（易语言打开数据库的数量不受限制，其数目仅受 Windows 操作系统限制），方便同时对多个数据库进行操作。

## 2) 置当前数据库

系统内部有一个“当前数据库”状态值，它被用来指向某一个已被打开的数据库。绝大部分数据库操作命令都针对当前数据库，比如：记录指针的移动、字段的读写等。如果同时打开两个或两个以上的数据库，在对某个数据库进行操作时，要使用“置当前库（）”命令。

命令原型为：

<逻辑型> 置当前库（文本型 数据库别名或名称）

“置当前库（）”命令可以改变系统中“当前数据库”状态值的指向，

例如：

置当前库（“学生表”）





3) 关闭数据库

在程序结束或在程序中不再使用某个数据库的时候，要养成关闭数据库的好习惯。

关闭数据库可以使用“关闭（）”命令。

命令原型为：

<无返回值> 关闭（[文本型 数据库别名或名称]）

“关闭（）”命令用来关闭已经打开的指定数据库。如果没有提供数据库名称，默认为关闭当前数据库。

如果一次性关闭已经打开的所有数据库，可以使用“全部关闭（）”命令。“全部关闭（）”命令无参数，该命令用来关闭当前已经打开的所有数据库。

例如：在启动窗口的“可否被关闭”事件里输入以下代码，则程序退出之前将关闭所有已打开的数据库。

子程序名	返回值类型	公开	备注
__启动窗口_可否被关闭	逻辑型		

全部关闭（）

5.7.2 数据库指针的跳转命令

在对记录操作前需要先了解记录指针的概念。

每一个被打开的数据库都有一个“当前记录指针”状态值，它指向数据库中的某一条记录，为一些记录读写命令提供位置指示，如读、写、删除等。它除了指向正常的记录外，还可能具有以下状态值之一。

1) “首记录前（）”命令

表明当前记录指针已经移动到了数据库首记录的前面，此时如果执行读写当前记录的命令肯定会失败，因为无法找到对应的记录读写位置。使用“首记录前（）”命令可以测试到此状态值。

2) “尾记录后（）”命令

表明当前记录指针已经移动到了数据库最后一条记录的后面，此时如果执行读写当前记录的命令也会失败。使用“尾记录后（）”命令可以测试到此状态值。

使用“取记录号（）”命令可以取回当前记录指针所指向记录的编号（从1开始）。如果为0，表示在首记录前；如果大于最大记录编号，表明在尾记录后。

3) “到首记录（）”命令

该命令可以将当前记录指针移动到数据库的首记录上。

4) “到尾记录（）”命令

该命令可以将当前记录指针移动到数据库的最后一条记录上。

第五章



### 5) “跳过 ()” 命令

该命令可以相对移动当前记录指针。通过移动当前记录指针，可以遍历数据库中的所有记录。如果参数被省略，默认值为 1，即向后移动一条记录。如果参数被设置为-1，则会从数据库尾记录到首记录，反向遍历记录。

根据这几个命令可以很容易地实现数据的遍历显示，在后面的代码中会多次用到这几个命令。

另外，在数据库的存在期间，同一数据库内任何时候任何一条记录（包括已经被实际删除的记录）都有一个唯一的非 0 正整数作为标记，称为“标签”，使用“标签”也可以实现记录的跳转，相关命令如下：

#### (1) “取标签 ()” 命令

使用本命令可以取回当前数据库当前记录的标签值。如果命令执行失败，将返回 0。

#### (2) “标签跳转 ()” 命令

参数为记录的标签值，标签跳转 () 命令可以改变当前数据库中的当前记录指针到具有指定标签的记录。如成功找到返回真，并且将当前记录指针移至该记录。如出错或未找到则返回假，当前记录指针位置保持不变。

#### (3) “记录是否存在 ()” 命令

参数为记录的标签值，如果当前数据库中存在具有指定标签值的记录，命令返回真，否则返回假。本命令可以用作在多用户环境中检查某记录是否被其他用户实际删除。

## 5.7.3 记录读取命令

这一部分，需要了解对易数据库记录的读取功能和数据源对表格的显示功能。

记录字段的读取，均在当前数据库的当前记录处进行，有以下几个命令。

#### 1) “读 ()” 命令

“读 ()” 命令返回当前数据库中当前记录处指定字段的数据内容。如“读 (#姓名)”可以返回当前记录处员工的姓名。

#### 2) “读字段 ()” 命令

可以读取非当前数据库内的记录字段。“读字段 ()” 命令和“读 ()” 命令的区别在于“读字段 ()” 命令可以读出非当前数据库的记录。

如“读字段 (#姓名, “学生表”)”可以返回“学生表”当前记录处学生的姓名。

读取数据库里全部记录的代码如下：

子程序名		返回值类型	公开	备注
_按钮_显示全部_被单击				
变量名	类型	静态	数组	备注
计次	整数型			



```

子程序_添加表头 ()
到首记录 ()
--▶ 计次循环首 (取记录数 (), 计次)
    子程序_读取当前记录 ()
    跳过 ()
--- 计次循环尾 ()
    
```

第一行代码表示为表格添加表头的子程序，将在介绍数据源常用命令时加以介绍。

第二行“到首记录（）”命令，表示将指针移动到首记录处。

第三行是一个循环，使用了“取记录数（）”命令。该命令会返回当前数据库的记录数目。

第四行是一个子程序。代码如下：

子程序名	返回值类型	公开	备注
子程序_读取当前记录			读取当前指针处的记录

```

数据源1.添加行 (1)
数据源1.置文本 (数据源1.取行数 (), 1, 读 (#姓名))
数据源1.置文本 (数据源1.取行数 (), 2, 选择 (读 (#性别), “男”, “女”))
数据源1.置文本 (数据源1.取行数 (), 3, 到文本 (读 (#年龄)))
数据源1.置文本 (数据源1.取行数 (), 4, 到文本 (读 (#出生日期)))
数据源1.置文本 (数据源1.取行数 (), 5, 到文本 (读 (#工资)))
数据源1.置文本 (数据源1.取行数 (), 6, 到文本 (读 (#奖金)))
数据源1.置对齐方式 (1, 1, 数据源1.取行数 (), 数据源1.取列数 (), #中中)
数据源1.加线条 (1, 1, 数据源1.取行数 (), 数据源1.取列数 (), 1 + 2 + 4 + 8 + 16 + 32)
    
```

该子程序使用了读（）命令，读取当前指针处记录的字段值。注意将读取的字段值显示到表格的时候，不是文本型的数据要转换到文本型。

第五行是跳过（）命令，表示每次读取完当前指针处记录之后，指针移动到后一条记录。通过这个循环可以读取数据库中所有的记录。

前面已讲解了记录指针的概念，这里应用记录指针，就可以轻松地实现“到首记录”、“到尾记录”、“上一条”、“下一条”功能，做到记录的逐条显示。

“到首记录”按钮的单击事件里输入代码如下：

子程序名	返回值类型	公开	备注
_按钮_到首记录_被单击			

```

到首记录 ()
子程序_添加表头 ()
子程序_读取当前记录 ()
    
```

子程序第一行代码表示指针跳转到第一条记录处。

第二行代码表示重新添加了表头。

第三行代码表示读取当前记录并显示在表格上。



“到尾记录”按钮的单击事件与“到首记录”按钮的单击事件类似，只是指针跳转到最后一条记录。

“上一条”按钮的单击事件里输入代码如下：

子程序名	返回值类型	公开	备注
_按钮_上一条_被单击			

```

跳过 (-1)
-- 如果真 (首记录前 ())
    信息框 (“已经到首记录前！”，0 + #信息图标, “提示”)
    返回 ()
子程序_添加表头 ()
子程序_读取当前记录 ()
    
```

可以看到，子程序的第一行代码表示指针跳转到上一条记录处。  
 第二行到第四行代码表示如果指针已经到了首记录之前，提示用户并返回。  
 第五行代码表示重新添加了表头。  
 第六行代码表示读取当前记录并显示在表格上。

“下一条”按钮的单击事件与“上一条”按钮的单击事件类似，只是指针跳转到后面一条记录。

### 5.7.4 数据源的常用命令

#### 1) 数据源的显示命令

在“子程序\_添加表头 ()”、“子程序\_读取当前记录 ()”子程序中都使用了数据源的命令。这里来介绍一下：

数据源常用命令包括：

- (1) 取行数 ()：返回数据源中现行数据行数。
- (2) 删除行 ()：在数据源中指定位置处删除数据行。成功返回真，失败返回假。
- (3) 插入行 ()：在数据源中指定位置处插入新数据行。

**注解：**某些数据提供者可能不支持在中间插入行。成功返回真，失败返回假。

- (4) 置文本 ()：设置数据源中指定单元格的文本内容，其参数描述见表5-9。

表5-9 参数描述

参数名	描 述
行号	必需的；整数型，行号从1开始
列号	必需的；整数型，列号从1开始
文本或图片文件名	必需的；文本型

- (5) 置对齐方式 ()：设置数据源中指定单元格在表现时所使用的对齐方式。

**注解：**如果数据源所使用的数据提供者不支持某些属性，该属性将被忽略。其参数描述见表5-10。



表5-10 参数描述

参数名	描 述
行号	必需的; 整数型, 行号从1开始
列号	必需的; 整数型, 列号从1开始
行数	整数型, 可以被省略。对齐方式应用范围的行数。如果本参数被省略, 默认值为1
列数	整数型, 可以被省略。对齐方式应用范围的列数。如果本参数被省略, 默认值为1
对齐方式	整数型, 可以被省略。对齐方式样式值。可以为以下常量值之一: 1 (#上左); 2 (#上中); 3 (#上右); 4 (#中左); 5 (#中中); 6 (#中右); 7 (#下左); 8 (#下中); 9 (#下右); 10 (#缩放图片); 11 (#居中图片); 12 (#缩放居中图片); 13 (#平铺图片); 14 (#缩放平铺图片)。其中 10 到 14 对齐方式仅对图片型单元格有效。如果本参数被省略, 默认值为“#上左”

(6) 加线条 ( ) : 为数据源中指定范围内单元格添加线条。

**注意:** 如果数据源所使用的数据提供者不支持此特性, 本命令将被忽略。其参数描述见表5-11。

表5-11 参数描述

参数名	描 述
行号	必需的; 整数型, 行号从1开始
列号	必需的; 整数型, 列号从1开始
行数	整数型, 可以被省略。欲添加线条的单元格范围行数。如果本参数被省略, 默认值为1
列数	整数型, 可以被省略。欲添加线条的单元格范围列数。如果本参数被省略, 默认值为1
线条类型	整数型, 可以被省略。欲添加线条的类型。可以为以下常量值之一或之和: 1 (#左边框); 2 (#上边框); 4 (#右边框); 8 (#下边框); 16 (#水平线); 32 (#垂直线); 64 (#单斜线); 128 (#双斜线); 256 (#交叉线)。如果本参数被省略, 默认值为“#左边框 + #上边框 + #右边框 + #下边框”

数据源的其他命令、事件详见“第二部分 易语言的命令与组件”中“4.12 数据库类组件”相关内容。

通过数据源命令的介绍, 来分析一下添加表头的子程序。代码如下:

子程序名	返回值类型	公开	备注
子程序_添加表头			

```
数据源1.删除行 (1, 数据源1.取行数 ())  
--- 如果真 (数据源1.插入行 (1, 1) = 假)  
    返回 0  
数据源1.置文本 (1, 1, “姓名”)  
数据源1.置文本 (1, 2, “性别”)  
数据源1.置文本 (1, 3, “年龄”)  
数据源1.置文本 (1, 4, “出生日期”)  
数据源1.置文本 (1, 5, “工资”)  
数据源1.置文本 (1, 6, “奖金”)  
数据源1.置对齐方式 (1, 1, 数据源1.取行数 (), 数据源1.取列数 (), #中中)  
数据源1.加线条 (1, 1, 数据源1.取行数 (), 数据源1.取列数 (), 1 + 2 + 4 + 8 + 16 + 32)
```

第一行代码表示为将表格清空。  
第二、三行表示插入一行, 如果失败则返回。  
第四行到第九行表示将字段名显示到表头中。  
第十行用来设置表头的对齐方式。





第十一行代码表示为表头添加线条，可以起到美观的效果。

继续来分析一下读取当前记录的子程序。代码如下：

子程序名	返回值类型	公开	备注
子程序_读取当前记录			读取当前指针处的记录

数据源1. 添加行 (1)

数据源1. 置文本 (数据源1. 取行数 0, 1, 读 (#姓名))

数据源1. 置文本 (数据源1. 取行数 0, 2, 选择 (读 (#性别), “男”, “女”))

数据源1. 置文本 (数据源1. 取行数 0, 3, 到文本 (读 (#年龄)))

数据源1. 置文本 (数据源1. 取行数 0, 4, 到文本 (读 (#出生日期)))

数据源1. 置文本 (数据源1. 取行数 0, 5, 到文本 (读 (#工资)))

数据源1. 置文本 (数据源1. 取行数 0, 6, 到文本 (读 (#奖金)))

数据源1. 置对齐方式 (1, 1, 数据源1. 取行数 0, 数据源1. 取列数 0, #中中)

数据源1. 加线条 (1, 1, 数据源1. 取行数 0, 数据源1. 取列数 0, 1 + 2 + 4 + 8 + 16 + 32)

第一行代码表示为表格添加一行。

第二行到第七行表示将数据库中的记录显示到表格中，使用了“置文本 ( )”命令。

“取行数 ( )”命令取得当前的数据源行数，用它来作为“置文本 ( )”的行参数。“读 ( )”命令读取到了当前指针处的字段值，作为“置文本 ( )”命令的第三个参数，表示欲显示的表格的内容。

**注解：**将读取的字段值显示到表格的时候，不是文本型的数据要转换到文本型。

第八行代码表示设置表格的对齐方式。

第九行代码表示为表格添加线条，可以起到美观的效果。

## 2) 数据源的打印方法

这一部分，将简单介绍数据源对打印机的设置，以实现“打印数据”功能。

如果不使用数据源，直接使用表格的“打印 ( )”、“打印预览 ( )”命令。

例如：

表格1. 打印 (真, “工资表”)

或

表格1. 打印预览 ( )

在默认情况下，很有可能会发现打印的数据不全或显示效果不好。实际上数据源对打印机的设置也提供了相关的命令，使用这些命令可以很好地控制打印的格式和效果。

相关的命令有：“打印设置 ( )”、“取打印设置 ( )”、“置打印设置 ( )”、“取打印页宽 ( )”、“取打印页高 ( )”。

这里讲一下“置打印设置 ( )”命令。

“置打印设置 ( )”命令设置打印数据源数据时将使用的设置信息。成功返回真，失败返回假。参数<1>的名称为“打印设置信息”，类型为“打印设置信息”。

“打印设置信息”数据类型的成员有：纸张类型、纸张方向、左边距、上边距、右边距、下边距、每页打印行数、打印缩放比等，用户在使用时可以自行设定。





例如：在“打印数据”按钮的单击事件中输入以下代码。

子程序名	返回值类型	公开	备注
_按钮_打印数据_被单击			

变量名	类型	静态	数组	备注
局部_打印配置	打印设置信息			

```
局部_打印配置.纸张方向 = 1
局部_打印配置.纸张类型 = #A4纸
局部_打印配置.每页打印行数 = 16
局部_打印配置.打印缩放比 = 100
局部_打印配置.上边距 = 150
局部_打印配置.左边距 = 40
局部_打印配置.右边距 = 40
数据源1.置打印设置 (局部_打印配置)
表格1.打印预览 ()
```

相应地设置了纸张方向、纸张类型、每页打印行数、打印缩放比、上边距、左边距、右边距等成员。

- 第一行到第七行代码表示对数据类型“打印设置信息”的设置。
- 第八行表示设置打印数据源数据时所将使用的设置信息。
- 第九行表示使用了表格的打印预览功能。用户可以在预览之后打印。

5.7.5 记录的添加与修改命令

这一部分，将介绍“添加数据”、“修改数据”功能，实现对易数据库记录的写操作。

添加记录有以下几个命令：

1) “加空记录 ()” 命令

“加空记录 ()” 命令可以在当前数据库的尾部添加一条新的空记录。加空记录之后，可以使用“写 ()” 或“写字段 ()” 命令。对当前新加的空记录进行写操作，“写 ()” 命令稍候介绍。

例如：

```
加空记录 ()
写 (#姓名, “田亮”)
```

2) “加记录 ()” 命令

“加记录 ()” 命令可以同时提供欲添加的数据。

例如：

```
加记录 (“王平”, 真, 31)
```

语句可以在当前数据库的尾部添加一条名为“王平”、性别为“真”、年龄为31 的新员工记录。





## 3) “添加 ()” 命令

“添加 ()” 命令可以将其他数据库内的记录添加到本数据库。

## 【例】

添加(“员工表2”,,)’ 可以将“员工表2.edb数据库中的所有记录添加到当前数据库的尾部

添加(“员工表2”,取记录号 () ≤ 5,)’ 语句仅可以添加前5条记录

添加(“员工表2”,是否已删除 () = 假,)’ 语句仅可以添加所有未被删除的记录

记录字段的写操作均在当前数据库的当前记录处进行,有以下几个命令。

## 4) “写 ()” 命令

“写 ()” 命令将数据写入到当前数据库中当前记录处的指定字段内。如“写(#姓名, “李敏”)” 可以将当前记录处员工的姓名写为“李敏”。

## 5) “写字段 ()” 命令

“写字段 ()” 命令可以写入非当前数据库内的记录字段。“写字段 ()” 命令和“写 ()” 命令的区别在于“写字段 ()” 命令可以写入非当前数据库的记录。如“写字段(#姓名, “员工表2”, “李敏”)” 可以将“员工表2” 当前记录处员工的姓名写为“李敏”。

## 6) “修改 ()” 命令

“修改 ()” 命令可以一次性修改当前记录的多个字段,参数的名称为“修改数据”,类型为“通用型”,本命令的每一个参数值用作替换当前记录中对应位置处的一个字段。如果某参数值被省略,则对应位置处字段的内容将保持不变。如:“修改(“易翔”,假,2)” 语句可以将当前记录的员工姓名改变为“易翔”,性别更改为:假,年龄改为2,其他字段不变。

“修改 ()” 命令与“写 ()” 命令、“写字段 ()” 命令的区别是该命令一次可以修改多个字段。

在“添加数据”按钮的单击事件里输入如下代码:

载入(窗口\_添加,,真)

载入“窗口\_添加”窗口,在该窗口中“添加”按钮的单击事件里输入如下代码(部分代码):

加空记录 0

如果(写字段(#姓名,,编辑框\_姓名.内容)且写字段(#性别,,选择(组合框1.现行选中项 = 0,真,假))且写字段(#年龄,,到整数(编辑框\_年龄.内容))且写字段(#出生日期,,取日期(日期框1.今天))且写字段(#工资,,到数值(编辑框\_工资.内容))且写字段(#奖金,,到数值(编辑框\_奖金.内容)))

信息框.(“添加记录成功”,0 + #信息图标,“成功”)

\_按钮\_显示全部\_被单击 0

销毁 0

信息框.(“添加记录失败”,0 + #信息图标,“失败”)





第一行代码表示将易数据库添加一条空记录，指针指向该记录处。

第二行到第五行表示将数据写入易数据库。

**注解：** 写入的数据要与易数据库的字段类型一致。

第六行表示写入数据成功提示。

第七行表示重新读取全部数据。

第八行表示销毁本窗口，返回主窗口。

第九行表示写入数据失败提示。

在“修改数据”按钮的单击事件里输入如下代码：

子程序名	返回值类型	公开	备注
_按钮_修改_被单击			

```
--- 如果真 (表格1.取光标行号 () ≤ 1)
    信息框 (“请选择要修改的记录。”， 0 + #警告图标, “警告”)
    返回 ()
跳到 (表格1.取光标行号 () - 1)
载入 (窗口_修改, , 真)
```

第一行到第三行代码中使用了表格的一个命令：“取光标行号（）”，该命令返回光标所在行的行号。如果返回的行号小于等于1，说明选择的是表头或未选择表数据，提示错误信息并返回。表格的命令、事件、属性将在“第二部分 易语言的命令与组件”中“4.11 显示类组件”详细介绍。

第四行代码使用了易数据库提供的“跳到（）”命令，将记录指针跳转到相应的记录上去。

第五行代码表示载入修改界面，修改界面载入后，首先将本条记录显示在修改界面中，使用的语句如下：

子程序名	返回值类型	公开	备注
_窗口_修改_创建完毕			

```
编辑框_姓名.内容 = 读 (#姓名)
组合框1.现行选中项 = 选择 (读 (#性别) = 真, 0, 1)
编辑框_年龄.内容 = 到文本 (读 (#年龄))
日期框1.今天 = 读 (#出生日期)
编辑框_工资.内容 = 到文本 (读 (#工资))
编辑框_奖金.内容 = 到文本 (读 (#奖金))
```

在“修改”按钮的单击事件里输入如下代码：

```
如果 (修改 (编辑框_姓名.内容, 选择 (组合框1.现行选中项 = 0, 真, 假), 到
整数 (编辑框_年龄.内容), 取日期 (日期框1.今天), 到数值 (编辑框_工资.内容
), 到数值 (编辑框_奖金.内容)) = 真)
    信息框 (“修改记录成功”， 0 + #信息图标, “成功”)
    _按钮_显示全部_被单击 ()
    销毁 ()
信息框 (“修改记录失败”， 0 + #信息图标, “失败”)
```





第一行到第三行表示修改字段，依次修改了数据库中的6个字段。

**注解：** 参数值和数据库中字段的位置要一一对应。

第四行表示修改数据成功提示。

第五行表示重新读取全部数据。

第六行表示销毁本窗口，返回主窗口。

第七行表示修改数据失败提示。

## 5.7.6 记录的删除命令

这一部分将介绍如何删除数据库中的记录。

删除记录有以下几个命令。

### 1) “删除 ()” 命令

命令原型为：

<逻辑型> 删除 ([条件语句型 记录范围])

“删除 ()” 命令将当前数据库中的当前记录或者指定范围内的记录打上删除标记或取消删除标记，当前记录指针位置保持不变。成功返回真，失败返回假。

参数<1>的名称为“记录范围”，类型为“条件语句型”，可以被省略。如果本参数被省略，默认为删除或恢复删除当前记录，否则删除或恢复删除所有满足此条件的记录。如果欲删除或恢复删除全部记录，可直接将逻辑值“真”赋予本参数。

此命令并不实际删除记录，仅仅只加上删除标记，记录依旧存在并可以读写。只有当执行“彻底删除 ()” 命令后，这些被加上删除标记的记录才会从数据库中真正清除。

### 2) “彻底删除 ()” 命令

命令原型为：

<逻辑型> 彻底删除 ()

“彻底删除 ()” 命令将当前数据库中所有被打上删除标记的记录从数据库中彻底清除，当前记录指针位置将被移动到数据库首记录。记录被彻底删除后将不复存在，并且无法恢复。

**注解：** 执行本命令将重组数据库记录，因此速度比较慢。成功返回真，失败返回假。

要想真正删除记录，一定要执行本命令。

在“删除数据”按钮的单击事件里，输入如下代码：

子程序名	返回值类型	公开	备注
按钮_删除_被单击			

变量名	类型	静态	数组	备注
姓名	文本型			

姓名 = 数据源1.取文本 (表格1.取光标行号 0, 1)





```

如果真 (信息框 (“确认删除姓名为:” + 姓名 + “的记录?”, 0 + #确认取消钮 +
#询问图标 + 256, “确认删除”) ≠ #确认钮)
    返回 0
跳到 (表格1.取光标行号 0 - 1)
删除 0
    如果 (彻底删除 0 = 真)
        信息框 (“记录删除成功。”, 0 + #信息图标, “成功”)
    信息框 (“记录删除失败。”, 0 + #错误图标, “失败”)
    按钮_显示全部_被单击 0
    
```

第一行表示取得欲删除的那一条记录的姓名。

第二行到第四行表示提示用户确认是否要删除该记录。用户点击确认则继续，否则返回。

第五行代码使用了易数据库提供的“跳到（）”命令，将记录指针跳转到相应的记录上去。

第六行表示为当前记录置删除标记。

第七行到第九行表示删除该记录，成功则提示删除成功信息，失败则提示删除失败信息。

第十行表示重新读取全部数据。

另外易语言还提供了其他的删除相关命令，有以下几个：

### 3) “清空（）”命令

命令原型为：

<逻辑型> 清空（）

“清空（）”命令彻底清除当前数据库中的所有记录。成功返回真，失败返回假。

### 4) “是否已删除（）”命令

命令原型为：

<逻辑型> 是否已删除（）

“是否已删除（）”命令判断本条记录是否已经打上删除标记，如果当前数据库的当前记录已经被“删除”命令加上删除标记，返回真，否则返回假。

### 5) “恢复删除”命令

命令原型为：

<逻辑型> 恢复删除（[条件语句型 记录范围]）

“恢复删除”命令去掉当前数据库中当前记录或者指定范围内记录的删除标记，当前记录指针位置保持不变。成功返回真，失败返回假。

参数<1>的名称为“记录范围”，类型为“条件语句型”，可以被省略。如果本参数被省略，默认为删除或恢复删除当前记录，否则删除或恢复删除所有满足此条件的记录。如果欲删除或恢复删除全部记录，可直接将逻辑值“真”赋予本参数。





如果想取消删除标记，可以使用此命令。

## 5.7.7 复制记录与复制结构命令

### 1) “复制记录 ()” 命令

命令原型为：

<逻辑型> 复制记录 (数据库文件名, [记录条件], [字段范围], ...)

“复制记录 ()” 命令可以复制当前数据库的记录到另外一个数据库。其参数描述见表5-12。

表5-12 参数描述

参数名	描 述
数据库文件名	必需的；文本型。本参数指定欲复制到的数据库文件，该数据库必须不存在或者尚未被打开。如果已经存在，将会被覆盖
记录条件	可选的；条件语句型。如果本参数被省略，默认为当前数据库中的所有记录，否则将仅限于满足此条件的记录
字段范围	可选的；可扩充的；文本型，参数数据可以同时提供数组或非数组数据。本参数的各参数值顺序指定所有欲复制字段的名称。如果被省略，默认为所有字段

例如：要将“员工表”中所有姓李的员工记录都拷贝到新数据库“员工表2.edb”中，程序代码如下：

复制记录 (“员工表2”, 读 (#姓名) ≈ “李”, )’ 符号使用 “?” 输入

复制后“员工表2.edb”中的记录如图5-28所示。

假如你只想把“员工表”中的“姓名”和“年龄”字段复制过去，就需要使用字段范围参数，程序代码如下：

复制记录 (“员工表2”, , #姓名, #出生日期)

复制后“员工表2.edb”中的记录，如图5-29所示。

图5-28 复制记录

图5-29 复制部分字段

如果想把限制条件和限制字段得到的结果同时复制过去，例如要求姓名为姓“李”，并且只要“姓名”和“年龄”字段，就要写成这样的格式：

复制记录 (“员工表2”, 读 (#姓名) ≈ “李”, #姓名, #年龄)





复制后“员工表2.edb”中的记录，如图5-30所示。

可以看到，结果为一条符合条件记录。

2) “复制结构 ()” 命令

命令原型为：

<逻辑型> 复制结构 (数据库文件名, [字段范围], ...)

“复制结构 ()” 命令可以复制当前数据库的结构到另外一个数据库。所谓的“结构”就是数据库字段的定义和顺序。其参数描述见表5-13。

表5-13 参数描述

参数名	描 述
数据库文件名	必需的；文本型。本参数指定欲复制到的数据库文件，该数据库必须不存在或者尚未被打开。如果已经存在，将会被覆盖
字段范围	可选的；可扩充的；文本型，参数数据可以同时提供数组或非数组数据。本参数的各参数值顺序指定所有欲复制字段的名称。如果被省略，默认为所有字段

如果想把“员工表”中的“姓名”、“工资”两个字段的结构复制给“员工表2”，可以输入如下代码：

复制结构 (“员工表2”, #姓名, #工资)

执行完该语句之后，使用菜单“数据库”→“结构编辑器”，打开“员工表2”数据库，如图5-31所示。

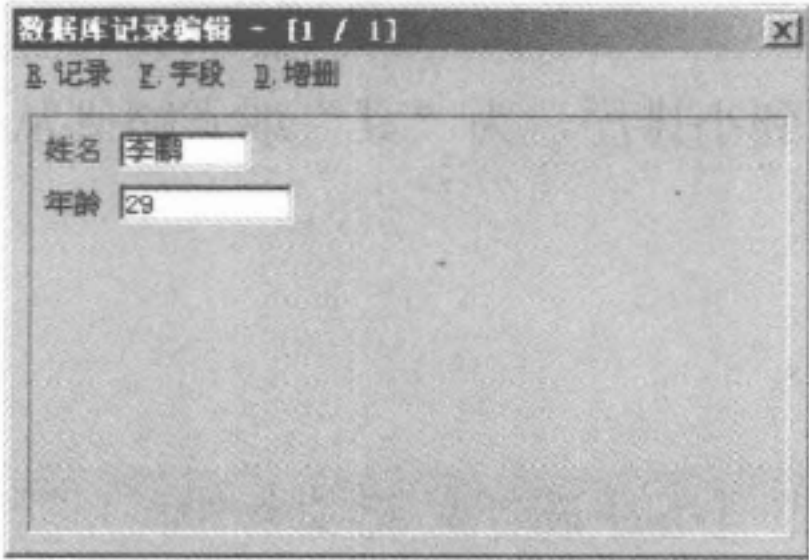


图5-30 复制部分字段

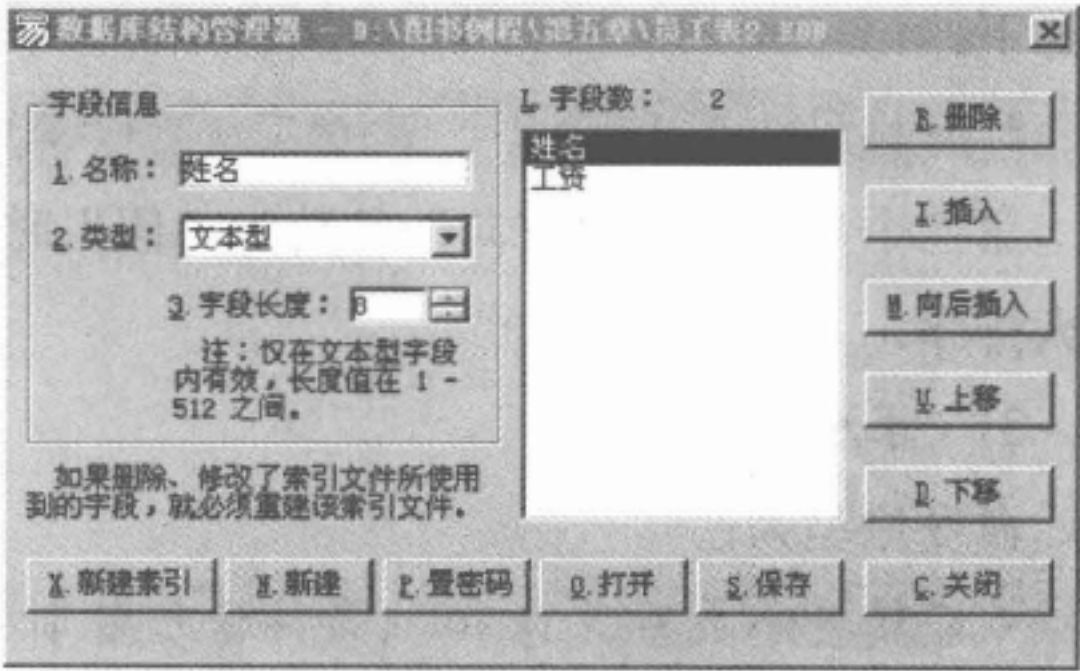


图5-31 复制结构命令的使用

“姓名”、“工资”两个字段的结构已经复制过去了。

5.7.8 数值统计类命令

易语言的数据库还提供了许多数值统计类的命令，下面来了解一下。

1) “排序 ()” 命令

命令原型为：

<逻辑型> 排序 (数据库文件名, 排序字段, [排序方向], [记录条件], [字段范围], ...)





“排序（）”命令根据指定字段排序复制当前数据库的记录到另外一个数据库。命令执行后当前记录指针保持不变。成功返回真，失败返回假。

例如：将“员工表1.edb”中的记录根据工资多少排序到“员工表2.edb”，代码如下：

排序（“员工表2”，#工资，假，，）

第三个参数“排序方向”参数为“假”时，为从大到小排序，为“真”或省略则从小到大排序。从记录编辑器打开数据库，可以看到的情况如图5-32所示。

姓名为“刘飞飞”的员工，工资最高，所以排在第一条记录。

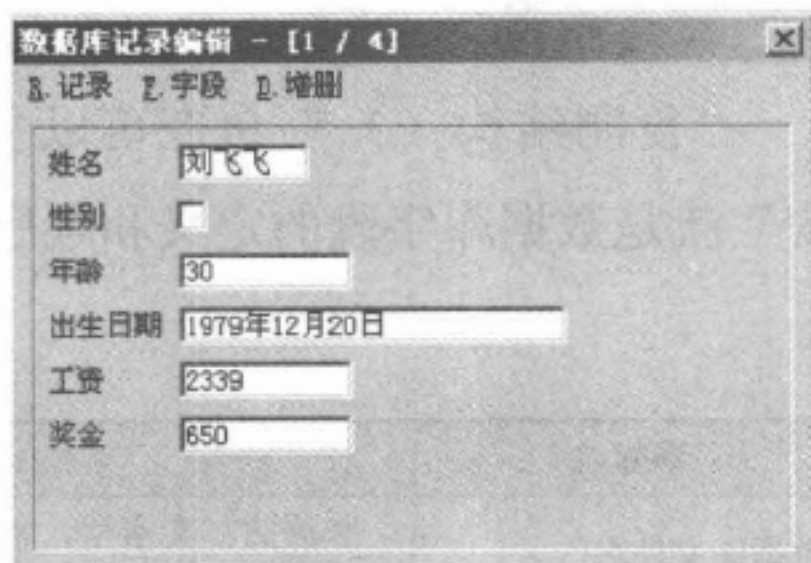


图5-32 排序命令的使用效果

## 2) “计算排序（）”命令

命令原型为：

<逻辑型> 计算排序（数据库文件名，排序表达式，[排序方向]，[记录条件]，[字段范围]，...）

“计算排序（）”命令根据指定数值表达式的计算值排序复制当前数据库的记录到另外一个数据库。命令执行后当前记录指针保持不变。成功返回真，失败返回假。

例如：将“员工表1.edb”中的记录根据工资加奖金的和排序到“员工表2.edb”，代码如下：

计算排序（“员工表2”，读（#工资）+ 读（#奖金），假，，）

第三个参数“排序方向”参数为“假”时，为从大到小排序，为“真”或省略则从小到大排序。

## 3) “求和（）”命令

命令原型为：

<双精度小数型> 求和（条件语句型 计算表达式，[条件语句型 记录条件]）

“求和（）”命令根据当前数据库计算并返回某数值型表达式的和，命令执行后当前记录指针保持不变。

例如：

局部\_总工资 = 求和（读（#工资）+ 读（#奖金），读（#姓名）= “李鹏”）

可以求出姓名为“李鹏”的员工的总工资。

## 4) “取平均值（）”命令

命令原型为：

<双精度小数型> 取平均值（条件语句型 计算表达式，[条件语句型 记录条件]）

“取平均值（）”命令根据当前数据库计算并返回某数值型表达式的平均值，命令执行后当前记录指针保持不变。





例如：

局部\_平均工资 = 取平均值(读(#工资),)

“取平均值”命令，可以将当前数据库中指定字段的每条记录的平均值取出，并且可以将当前数据库某个字段中，符合一定条件的记录的平均值取出。

例如：

局部\_平均工资 = 取平均值(读(#工资), 读(#工资)>1000)

上面的取平均值命令是将所有工资大于1000的记录的平均值取出。

#### 5) “取最大值 ()” 命令

命令原型为：

<双精度小数型> 取最大值 (条件语句型 计算表达式, [条件语句型 记录条件])

“取最大值 ()”命令根据当前数据库计算并返回某数值型表达式的最大值，当前记录指针被移动到具有最大值的记录。

例如：计算工资加奖金的和为最大的员工，并跳到该条记录处，代码如下：

取最大值(读(#工资) + 读(#奖金),)

可以声明一个双精度小数型变量“局部\_最大值”，用来接收该返回值：

局部\_最大值 = 取最大值(读(#工资) + 读(#奖金),)

运行后，“局部\_最大值”变量的结果为：“2689”，是姓名为：“刘飞飞”的员工的工资与奖金的和，该数值为当前员工中工资与奖金和的最大值。

#### 6) “取最小值 ()” 命令

命令原型为：

<双精度小数型> 取最小值 (条件语句型 计算表达式, [条件语句型 记录条件])

“取最小值 ()”命令根据当前数据库计算并返回某数值型表达式的最小值，当前记录指针被移动到具有最小值的记录。

例如：计算工资加奖金的和为最小的员工，并跳到该条记录处，代码如下：

局部\_最小值 = 取最小值(读(#工资) + 读(#奖金),)

运行后，局部\_最小值的结果为：“1335”，是姓名为：“李鹏”的员工的工资与奖金的和，为当前员工中工资与奖金和的最小值。

### 5.7.9 记录的查找

易数据库提供了对记录数据的查找功能，相关命令如下。

“查找 ()”命令的原型为：

<逻辑型> 查找 (条件语句型 查找条件)

“查找 ()”命令从当前数据库中当前记录位置处（包括当前记录）开始寻找符合给定条件的记录。如成功找到返回真，并且将当前记录指针移至所找到的记录。如出错或未





找到则返回假，当前记录指针位置保持不变。

参数<1>的名称为“查找条件”，类型为“条件语句型”。参数值指定欲查找记录应该满足的条件。

例如：

查找（读（#姓名）≈“李”）

表示查找易数据库中姓“李”的员工。

参看以下例程中的代码：

子程序名	返回值类型	公开	备注
_按钮_查找_被单击			

变量名	类型	静态	数组	备注
欲寻找姓名	文本型			

欲寻找姓名 = 删首尾空 (编辑框\_姓名.内容)

--- 如果真 (欲寻找姓名 = “”)

信息框 (“请输入欲寻找员工的姓名!”, #错误图标, “错误”)

编辑框\_姓名.获取焦点 ()

返回 ()

↓ 到首记录 ()

--- 如果真 (查找 (读 (#姓名) ≈ 欲寻找姓名) = 假)

信息框 (“未找到指定员工!”, #信息图标, )

返回 ()

↓ 子程序\_添加表头 ()

子程序\_读取当前记录 ()

子程序的第一行代码表示获取欲查找姓名的关键字。

第二行到第五行代码表示如果用户未填写欲查找关键字，提示错误并返回。

第六行表示将指针移动到首记录处，从首记录开始查找。

第七行到第九行代码使用了“查找（）”命令，查找当前数据库中姓氏与输入的关键字相同的员工，如果没有找到则提示并返回。

第十行、第十一行代码表示添加表头并读取当前记录显示到表格上。

“继续查找”功能的实现，实际上就是在当前指针处继续使用“查找（）命令”查找。“继续查找”按钮的单击事件输入代码如下：

子程序名	返回值类型	公开	备注
_按钮_继续查找_被单击			

变量名	类型	静态	数组	备注
欲寻找姓名	文本型			

欲寻找姓名 = 删首尾空 (编辑框\_姓名.内容)

--- 如果真 (欲寻找姓名 = “”)

信息框 (“请输入欲寻找员工的姓名!”, #错误图标, “错误”)

编辑框\_姓名.获取焦点 ()

返回 ()

↓ 跳过 ()





```

-- 如果真 (查找 (读 (#姓名) ~ 欲寻找姓名) = 假)
  信息框 (“未找到下一个指定员工!”, #信息图标, )
  到首记录 0
  返回 0
子程序_添加表头 0
子程序_读取当前记录 0

```

子程序的前五行与“查找”按钮的单击事件相同。

第六行跳过（）命令表示将指针向后移动一位，在此位置开始查找。

第七行到第十行代码表示执行“查找”命令查找出当前数据库中姓氏与输入的关键字相同的员工，如果没有查找到，将指针移动到首记录处，并返回。

第十一行、第十二行代码表示添加表头并读取当前记录显示在表格上。

“查找（）”命令与易语言系统支持库中的拼音处理类命令结合，可以解决常见的汉字近音搜寻问题。

例如：

```

查找 (发音比较 (“李鹏”, 删全部空 (读 (#姓名)), 真))
查找 (输入字比较 (“lipeng”, 删全部空 (读 (#姓名)), 真,))
查找 (输入字比较 (“lp”, 删全部空 (读 (#姓名)), 真,))

```

语句中的“删全部空（）”命令用作预先去除字段数据中可能存在的全半角空格，如果确定所有记录的该字段数据中都不存在空格，可以去掉此命令。

第一条语句可以查找出当前数据库中姓名发音为“李鹏”的员工，如：“李鹏”、“理朋”，“利碰”等；第二条语句完成类似的工作，不过使用的是全拼拼音编码；第三条语句使用的则是首拼拼音编码。

易语言的拼音处理类命令详见“第二部分 易语言的命令与组件”中“3.8 拼音处理命令”相关内容。

## 5.7.10 索引的创建与使用

易语言还提供索引查找功能，使用索引查找可以加快查找记录的速度。

“索引”的概念：索引是一个单独的、物理的数据库结构，它是某个表中一行或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。表的存储由两部分组成，一部分用来存放数据页面，另一部分存放索引页面。

通常，索引页面相对于数据页面来说小得多。数据检索花费的大部分时间是磁盘读写，没有索引就需要从磁盘上读表的每一个数据页，如果有索引，则只需查找索引页面就可以了。所以建立合理的索引，就能加速数据的检索过程。

**注意：**索引的存在也会降低对数据库记录的更新速度，所以除非是较大的数据库（最起码在一千条记录以上），否则一般不使用索引。





扩展名为“.enx”的文件是易数据库索引文件。索引文件由用户根据需要自行创建。

### 1) 索引的创建

索引的创建方法有菜单创建和代码创建两种。

菜单创建方法为：选择菜单“数据库”→“结构编辑器”，如图5-33所示。

点击按钮“新建索引”，如图5-34所示。

可以选择索引的字段，点击“创建”按钮，输入索引的名称，一个索引就创建好了，如图5-35所示。

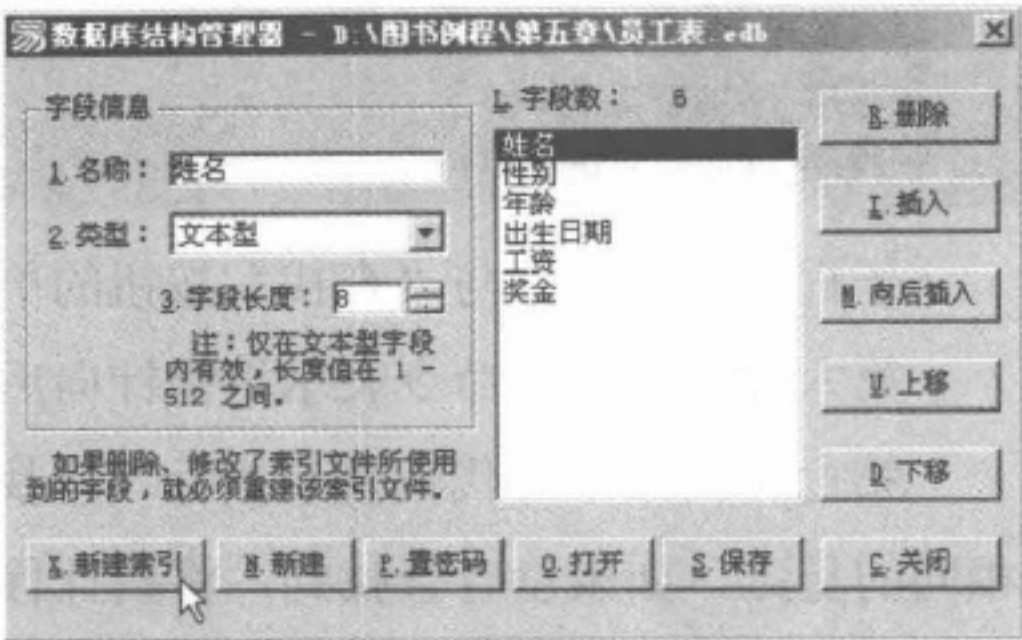


图5-33 选择“结构编辑器”

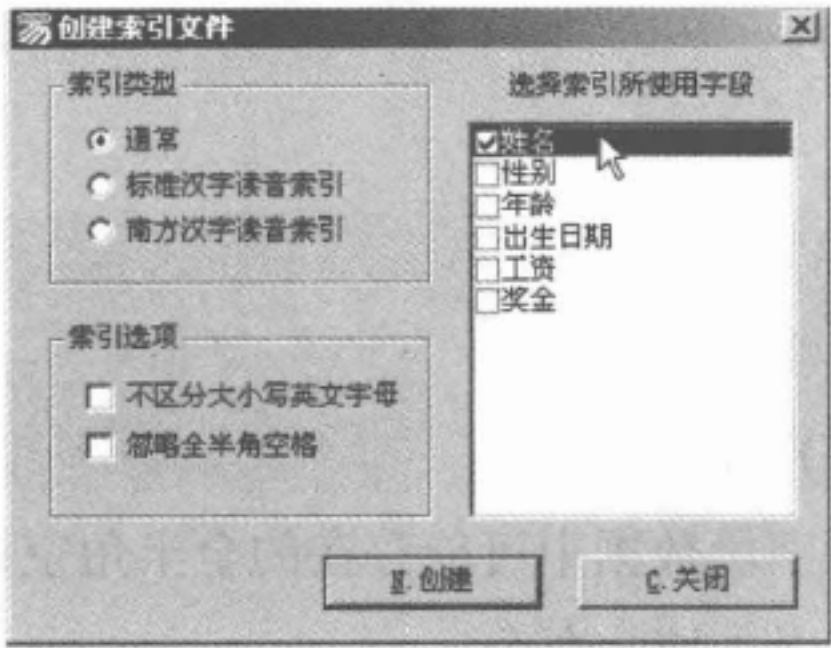


图5-34 选择“新建索引”

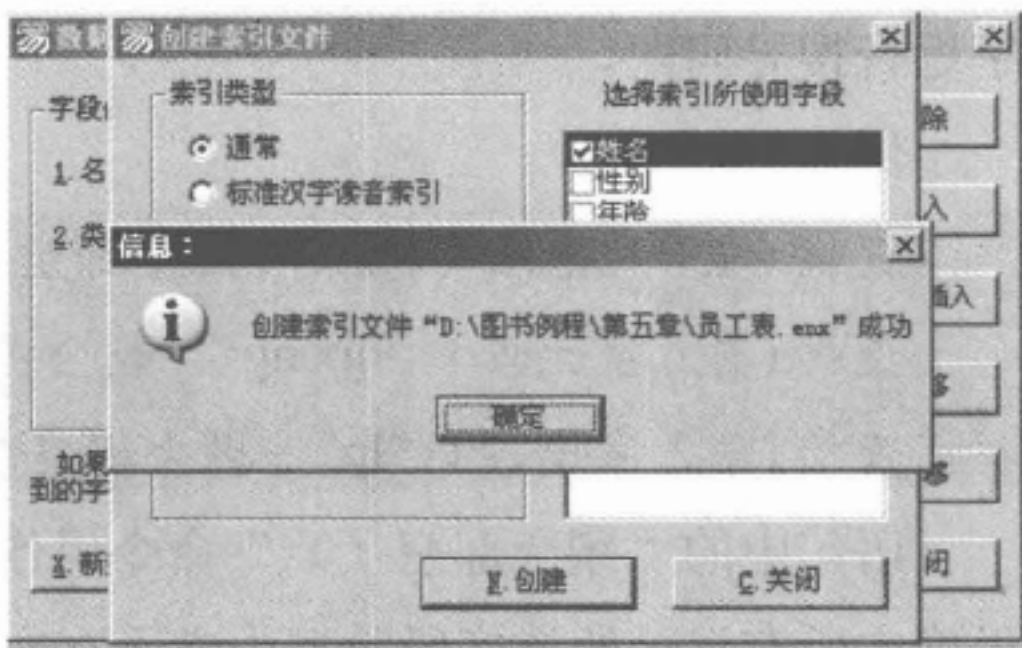


图5-35 创建索引成功

也可以在程序中使用代码创建，需要用到“新建索引（）命令”。

命令原型为：

<逻辑型> 新建索引（文本型 欲创建索引文件的名称，[整数型 索引类型]，[整数型 索引选项]，[整数型 索引块尺寸]，文本型数组/非数组 被索引字段的名称，...）

“新建索引”命令在当前数据库中创建并打开一个索引文件。成功返回真，失败返回假。命令参数表中最后一个参数可以被重复添加。其参数描述见表5-14。

表5-14 参数描述

参数名	描 述
欲创建索引文件的名称	必需的；文本型
索引类型	整数型, 可以被省略 参数值可以为以下常量值之一： 1 (#通常索引)； 2 (#读音索引)：在文本型索引字段中，所有汉字根据标准读音建立索引 3 (#南方读音索引)：在文本型索引字段中，所有汉字根据南方读音建立索引（南方读音不区分卷舌音和平舌音） 如果被省略，默认为“#通常索引”
索引选项	整数型, 可以被省略 参数值可以为以下常量值之一或者之和： 1 (#不区分大小写)：文本型索引字段中不区分英文字母的大小写 2 (#忽略所有空格)：文本型索引字段中忽略所有的全、半角空格 如果本参数被省略，默认为 0



续表

参数名	描 述
索引块尺寸	整数型, 可以被省略 参数值指定索引文件记录索引项时所使用索引块的大小, 以 512 字节为单位, 最大值为 10 如果本参数被省略, 默认值为 1。仅在大型数据库中需要建立大索引块的索引文件, 一般情况下使用默认值即可
被索引字段的名称	必需的; 文本型, 提供参数数据时可以同时提供数组或非数组数据。本参数的各参数值顺序指定所有被索引字段的名称, 支持以数组方式直接提供多个字段

在使用代码创建索引之前要先打开数据库。

例如:

```
打开 ("员工表.edb" , , , , , )  
新建索引 ("员工表" , , , , #姓名)
```

上面的语句建立了一个名为“员工表.enx”的索引文件, 索引字段为“姓名”字段, 也可以建立基于多个字段的索引文件。

索引只有被打开后才能被使用。使用“新建索引 ()”命令新建的索引会被自动打开并设置为当前索引。对于已经存在的索引文件, 必须在打开数据库时同步打开。

例如:

```
打开 ("员工表" , , , , , "员工表")
```

此语句同步打开了“员工表.enx”数据库索引文件, 并将其设置为当前索引。

2) 索引查找

索引查找 () 命令

命令原型为:

```
<逻辑型> 索引查找 (通用型 欲搜寻值, ...)
```

“索引查找 ()”命令在当前数据库中使用其当前索引来快速寻找某一记录, 寻找从当前记录位置 (包括当前记录) 处开始。注意为命令所提供的欲搜寻值参数的数目必须与被索引字段的数目一致。如成功找到返回真, 并且将当前记录指针移至所找到的记录。如出错或未找到则返回假, 当前记录指针位置保持不变。命令参数表中最后一个参数可以被重复添加。

参数<1>的名称为“欲搜寻值”, 类型为“通用型”。本参数数据的类型必须与索引字段表中对应位置处字段的类型一致。

“索引查找 ()”命令与“查找 ()”命令的使用方法是一样的。如果建立了索引, 使用索引查找前面的程序中编辑框中的内容, 对应的代码如下:

子程序名	返回值类型	公开	备注
按钮_打开_被单击			

```
如果 (打开 (#员工表, "员工表" , , , , "员工表") = 假)  
    信息框 ("员工数据库打开失败!", #错误图标, "错误")  
结束  
信息框 ("员工数据库打开成功!", #信息图标, "成功")
```



在打开数据库时需要同时打开索引。

子程序名	返回值类型	公开	备注
_按钮_查找_被单击			

变量名	类型	静态	数组	备注
欲寻找姓名	文本型			

```

欲寻找姓名 = 删首尾空 (编辑框_姓名.内容)
--- 如果真 (欲寻找姓名 = "")
    信息框 ("请输入欲寻找员工的姓名!", #错误图标, "错误")
    编辑框_姓名.获取焦点 ()
    返回 ()
到首记录 ()
--- 如果真 (索引查找 (欲寻找姓名) = 假)
    信息框 ("未找到指定员工!", #信息图标, )
    返回 ()
子程序_添加表头 ()
子程序_读取当前记录 ()
    
```

上面的语句只是在第七行由“查找（）”命令改为了“索引查找（）”命令。

现在必须要输入完整的姓名，如“李鹏”，才能显示李鹏这条记录，也就是说索引查找的参数必须与现实值完全一致。这一点“索引查找（）”命令没有“查找（）”命令方便。

索引也支持近似音索引文件。如：

```

新建索引 ("员工表", #读音索引, #不区分大小写 + #忽略所有空格, , #姓名)
新建索引 ("员工表", #南方读音索引, #不区分大小写 + #忽略所有空格, , #姓名)
    
```

以上两条语句分别建立一个基于标准读音和一个基于南方读音的索引文件。这样使用发音索引文件就可进行近似音搜寻。

例如：

```

“索引查找 (“里朋”)”
就可以找到 “李鹏”。
```

在打开数据库时应该使用其“索引文件表”参数同时打开所有的索引文件，以便索引文件能够得到及时的更新。也可以使用“更新索引（）”命令来强制更新当前索引文件。

通过以上的讲解，易数据库例程编写完成，相信大家已经掌握了如何使用易数据库。

## 5.8 我的播放器（五）

在第五章中，学习了易语言核心支持库中的易语言数据库部分。接下来，将易数据库运用到MP3播放程序中，制作一个与易数据库相联系的MP3播放器。



在前一章中，增加了对播放列表的操作，将播放列表保存成了一个文本文件，在本程序中除了要实现控制播放状态的功能，还要将播放列表换成数据库的形式保存。添加组件完成这些功能，可以分五步来编写代码。

第一步：设计界面。

新建一个易语言程序，设计程序界面。加入按钮、标签、进度条、列表框等组件，分别设置各组件的属性；将“启动窗口”的标题改为“我的播放器（五）”。

按钮1标题设置为：播放，按钮1的名称设置为：播放按钮；用作播放MP3文件；

按钮2标题设置为：暂停，按钮2的名称设置为：暂停按钮；用作暂停播放MP3文件；

按钮3标题设置为：继续，按钮3的名称设置为：继续按钮；用作继续播放MP3文件；

按钮4标题设置为：添加，按钮4的名称设置为：添加按钮；用作向播放列表框中加入要播放的MP3文件名（全路径），并将MP3文件名加入到数据库中；

按钮5标题设置为：删除，按钮5的名称设置为：删除按钮；用作删除播放列表框中的MP3文件名，并将MP3文件名从数据库中删除；

列表框1名称设置为：播放列表框；用作显示要播放的MP3文件名；

进度条1名称设置为：播放进度条；用作显示播放进度；

标签1标题设置为空；标签1可视设置为假；标签1名称设置为：进度标签；用作反馈播放进度。

调整各组件的大小和位置，界面效果如图5-36所示。

界面设计完成，下面来编写代码。

第二步：建立播放列表数据库。

既然要与易数据库相关联，那么先要创建易数据库。双击“启动窗口”进入代码编辑区。



图5-36 播放器界面效果

首先要判断存放播放列表的数据库文件是否存在，如果不存在就先创建它。创建数据库需要用到字段信息数据类型，而且字段信息必须是数组型。因只保存MP3的文件名，所以定义了只有一个成员的字段信息表数组，即只有一个字段。如果数据库已存在，就“打开”数据库，读取所有记录，显示在“播放列表框”中。如下代码所示：

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
字段信息表	字段信息		1	



```

--- 如果真 (文件是否存在 (取运行目录 () + "\mp3.edb") = 假)
    字段信息表 [1].名称 = "mp3文件名"
    字段信息表 [1].类型 = #文本型
    字段信息表 [1].最大文本长度 = 500
    创建 (取运行目录 () + "\mp3.edb", 字段信息表)
↓
打开 (取运行目录 () + "\mp3.edb", , , , , )
置当前库 (取运行目录 () + "\mp3.edb")
到首记录 ()
---▶ 计次循环首 (取记录数 (), )
    播放列表框.加入项目 (读 (1), )
    跳过 ()
--- 计次循环尾 ()
    
```

在播放器程序结束，即“启动窗口”被关闭销毁，要把打开的数据库关闭。如下代码所示：

子程序名	返回值类型	公开	备注
__启动窗口_将被销毁			

关闭 ()

第三步：添加删除播放列表。

对于播放列表的操作与前面的播放器程序是不一样的。

使用系统处理类命令中的“多文件对话框”命令打开目录选择MP3文件，将选择的MP3文件名存放到“文本组变量”中，并用“计次循环首”命令加入到“播放列表框”中。在加入的同时，还要将MP3文件名加入到数据库中。如下代码所示：

子程序名	返回值类型	公开	备注
_添加按钮_被单击			

变量名	类型	静态	数组	备注
文本组变量	文本型		0	
计次变量	整数型			

```

文本组变量 = 多文件对话框 ("添加mp3文件", "mp3文件 (*.mp3) |*.mp3", , , 真)
--- 如果真 (取数组成员数 (文本组变量) = 0)
    返回 ()
---▶ 计次循环首 (取数组成员数 (文本组变量), 计次变量)
    播放列表框.加入项目 (文本组变量 [计次变量], )
    加记录 (文本组变量 [计次变量])
--- 计次循环尾 ()
    
```

思考：用“写”命令向数据库写入记录该怎样写代码？

不想播放的MP3文件，就从“播放列表框”中将其删除。首先要删除数据库中的记录，数据库的记录号从1开始，而“播放列表框”的现行选中项目从0开始，所以在转换时要注意。如下代码所示：





子程序名	返回值类型	公开	备注
删除按钮_被单击			

```
--- 如果真 (播放列表框.现行选中项 ≠ -1)
    跳到 (播放列表框.现行选中项 + 1)
    删除 ()
    彻底删除 ()
    播放列表框.删除项目 (播放列表框.现行选中项)
```

思考：如果先删除“播放列表框”中的项目，结果会怎样？

第四步：播放MP3文件，反馈播放进度。

有了播放列表，就可以播放MP3文件了。播放状态的代码与前面编写的播放器程序的代码是一样。如下代码所示：

子程序名	返回值类型	公开	备注
播放按钮_被单击			

```
--- 如果真 (播放列表框.现行选中项 = -1)
    播放列表框.现行选中项 = 0
    同步播放MP3 (播放列表框.取项目文本 (播放列表框.现行选中项), , 进度标签, )
```

MP3文件播放进度反馈。如下代码所示：

子程序名	返回值类型	公开	备 注		
_进度标签_反馈事件	整数型				
参数名	类 型	参考	可空	数组	备 注
参数一	整数型				
参数二	整数型				

```
播放进度条.位置 = 参数一
--- 如果真 (参数一 = 100)
    --- 如果 (播放列表框.现行选中项 = 播放列表框.取项目数 () - 1)
        播放列表框.选择项目 (0, )
        播放列表框.选择项目 (播放列表框.现行选中项 + 1, )
    播放按钮_被单击 ()
```

第五步：控制播放状态。

双击“暂停按钮”和“继续按钮”，实现暂停播放和继续播放。代码如下所示：

子程序名	返回值类型	公开	备注
暂停按钮_被单击			

暂停播放MP3 ()

子程序名	返回值类型	公开	备注
继续按钮_被单击			

继续播放MP3 ()





至此，“我的播放器”系列程序就基本编写完成。从第一程序的一行代码，到如今的几十行代码，应用众多的组件和命令，甚至应用到易数据库。当然，一切都还没有结束，“我的播放器”系列程序还可以继续扩展。如：可以播放其他类型的声音文件或音频文件，演示音频波段，展示歌词等功能。这就需要继续加深学习易语言。后面将会介绍易语言对外部数据库的操作，以及易语言对外部编程资源的应用。让我们来继续学习吧！

## 5.9 小结

数据库是按照一定的数据结构来组织、存储和管理数据的仓库，随着信息技术和市场的发展，特别是20世纪90年代以后，数据库的使用更加广泛。易语言拥有完全自主知识产权和核心技术的数据库：“易数据库”。易数据库是一个小型的数据库管理系统，它以一种简单的、类似表格的形式组织信息，并形成持久化存储。

本章通过一个完整例程的编写，讲解了如何创建与操作易数据库，并介绍了易数据库相关的组件。要深入了解易数据库中“指针”的概念，在对易数据库记录操作的过程中要时刻注意指针所在的位置。

## 5.10 习题

5-1 扩展名为“\_\_\_\_\_”的文件是易数据库主文件。

5-2 扩展名为“\_\_\_\_\_”的文件是易数据库辅助数据文件，仅在数据库中有\_\_\_\_\_或者\_\_\_\_\_型字段时才存在，文件名称除了后缀外与数据库主文件相同，它必须与\_\_\_\_\_文件放在同一目录中。


5-3 扩展名为“\_\_\_\_\_”的文件是易数据库索引文件。索引文件由用户根据需要自行创建，使用它可加快查找记录的速度。

5-4 数据库在使用前需要先打开，欲打开指定数据库，请使用“\_\_\_\_\_”命令。

5-5 欲一次性关闭已打开的所有数据库，请使用“\_\_\_\_\_”命令。

5-6 试着自己编写一个简单的客户表数据库，熟练掌握添加、修改和删除记录的方法。





## 第六章 外部数据库的应用

### 本章目标

在本章结束时，我们能够：

- 了解什么是ODBC
- 了解什么是ADO
- 了解主流的外部数据库种类
- 学会使用ODBC方式操作外部数据库
- 学会使用ADO方式操作外部数据库
- 学会使用简单的SQL语句





## 6.1 外部数据库简介

前面介绍了易数据库的用法，但是在编程中必须要使用其他类型数据库的时候该怎么办？易语言也提供了对易数据库之外的其他数据库的支持。

易语言可支持以ODBC方式和ADO方式操作外部数据库，比如Access数据库、dBase数据库、Sqlite数据库、Visual FoxPro数据库、MS SQL Server数据库、MYSQL数据库、Oracle数据库、SYBase数据库、PostgreSQL数据库等。

下面将对易语言外部数据库的调用与操作进行详细的介绍。

### 6.1.1 外部数据库组件

易语言5.1版推出了核心支持库中的“外部数据库”组件、“外部数据提供者”组件；数据库操作支持库中的“数据库连接”组件、“记录集”组件；MYSQL支持库；Sqlite数据库支持库。

“外部数据库”组件和“外部数据提供者”组件，封装了以ODBC方式访问数据库的标准接口，可以直接将外部数据库绑定到ODBC数据源，并对外部数据库进行操作。

“数据库连接”组件和“记录集”组件，封装了以ADO方式访问数据库的标准接口，使用ADO方式对外部数据库进行操作。

Mysql支持库可以对Mysql数据库系统进行操作。Mysql数据库是一个小型关系型数据库管理系统。Mysql的执行性能高，运行速度快，操作非常简单，并且还支持Linux操作系统。由于Mysql数据库为开放源代码数据库系统，较为安全，并且功能强大，因此在易语言中对Mysql数据库进行了直接的访问支持（不通过任何数据库协议），速度较快，易于安装与发布，是理想的网络数据库开发工具，并且由于它也支持Linux操作系统平台，因此成为跨平台的数据库首选工具。

Sqlite数据库支持库可以对Sqlite数据库进行操作。Sqlite数据库是一个小型关系型数据库；跨平台；支持SQL语句、事务、触发器、视图；速度相当快；小巧且不依赖任何额外的驱动程序。Sqlite数据库是本地数据库，不是网络数据库。相对于易语言数据库，Sqlite数据库的优势是支持SQL语句、事务、触发器、视图；相对于Microsoft Access，Sqlite数据库的优势是跨平台、无需额外驱动；相对于其他非本地数据库，如Oracle、DB2、MS SQL Server、MySQL、PostgreSQL，Sqlite数据库具有小巧、速度快的优势。

本章主要介绍“外部数据库”组件和“外部数据提供者”组件，“数据库连接”组件和“记录集”组件的用法。





## 6.1.2 ODBC与ADO

### 6.1.2.1 ODBC简介

ODBC (Open Database Connectivity, 开放数据库互连) 是微软公司开放服务结构中有关数据库的一个组成部分, 它建立了一组规范, 并提供了一组对数据库访问的标准API (应用程序编程接口)。这些API利用SQL语言来完成其大部分任务。ODBC本身也提供了对SQL语言的支持, 用户可以直接将SQL语句送给ODBC。

ODBC的最大优点是能以统一的方式处理所有的数据库。

应用程序要访问一个数据库, 首先必须用ODBC管理器注册一个数据源, 管理器根据数据源提供的数据库位置、数据库类型及ODBC驱动程序等信息, 建立起ODBC与具体数据库的联系。这样, 只要应用程序将数据源名提供给ODBC, ODBC就能建立起与相应数据库的连接。

ODBC不能直接访问数据库, 必须通过驱动程序管理器与数据库交换信息。驱动程序管理器负责将应用程序对ODBC的调用传递给正确的驱动程序, 而驱动程序在执行完相应的操作后, 将结果通过驱动程序管理器返回给应用程序。即客户应用程序连接ODBC数据源, ODBC数据源通过ODBC驱动程序管理与特定的ODBC驱动程序联系起来, 然后通过此ODBC驱动程序来访问本地或远程数据库。

ODBC数据源又叫DSN, 它把客户应用程序所要使用的驱动程序、数据库、用户名和口令等信息组合起来, 供客户端程序使用。

一个DSN可以定义为以下3种类型中的任意一种: ①用户数据源: 只能被所在计算机中的创建该数据源的用户使用; ②系统数据源: 所在计算机中任意用户, 只要拥有适当的权限都可以访问这个数据源; ③文件数据源: 可以在任意计算机中, 安装了合适的驱动程序的用户使用。

易语言在使用ODBC组件时, 就需要用到文件DSN。在使用数据源时, 首先要对数据源进行配置, 后面将会介绍如何配置ODBC数据源, 以及如何创建一个文件DSN。

配置ODBC数据源, 可以使用“控制面板”→“管理工具”→“数据源 (ODBC)”工具 (Windows XP系统)。

易语言中使用“外部数据库”组件实现以ODBC方式操作外部数据库。

### 6.1.2.2 ADO简介

ADO (ActiveX Data Object) 是微软公司为数据访问范例OLE DB而设计的, 是一个用于存取数据源的COM组件, 它提供了一个简洁的对OLE DB封装接口, 是一种面向对象的编程接口, 是编程语言和统一数据访问方式OLE DB的一个中间层, 是一个便于使用的应用程序层接口, 可以为任何数据源提供高性能的访问。允许开发人员编写访问数据的代码而不用关心数据库是如何实现的, 只需要关心到数据库的连接。





ADO是对当前微软所支持的数据库进行操作的最有效和最简单直接的命令，它是一种功能强大的数据访问编程模式，从而使得大部分数据源可编程的属性得以直接扩展。

易语言中使用“数据库连接”和“记录集”组件实现以ADO方式操作外部数据库。

ADO方式操作数据库比ODBC方式操作数据库效率更高。

### 6.1.3 Access数据库

目前，可供使用的数据库有很多，大型的数据库像MS SQL Server, Oracle, SYBase, DB2, MYSQL, PostgreSQL等，小型的数据库像FoxPro, Sqlite, Access等，在工作中经常会接触到它们，开发软件时也经常需要使用它们。下面以最常见的Access数据库为例，来介绍易语言对外部数据库的操作。

关系型数据库以行和列的形式存储数据，这一系列的行和列被称为表，多个表组成了数据库，多个数据库就组成了数据库系统。易数据库没有表的概念，或者说易数据库就是一个表。与易数据库不同，易语言外部的数据库大多数都有多个表，那些大型的数据库可以称为数据库系统，因为它们都可以管理多个数据库。

Access是微软Office软件包中的一个组件，它具有界面友好、易学易用、开发简单、接口灵活等特点，是典型的桌面数据库管理系统。以Access数据库为例，来建立一个简单的数据库。建立的数据库后面将会用到。

Access数据库的建立可以分成两个阶段，第一个阶段是根据要输入的数据性质，新增表并设置表的字段名称、数据类型；第二个阶段是在表内输入数据。

下面就以建立数据库（图书管理.mdb）中的“图书库存表”为例说明如何建立Access数据库。

第一步：打开Microsoft Access 2003，然后点击工具条中的“新建”按钮，或菜单“文件→新建”，在右边的窗口中选择“空数据库...”选项，如图6-1所示。

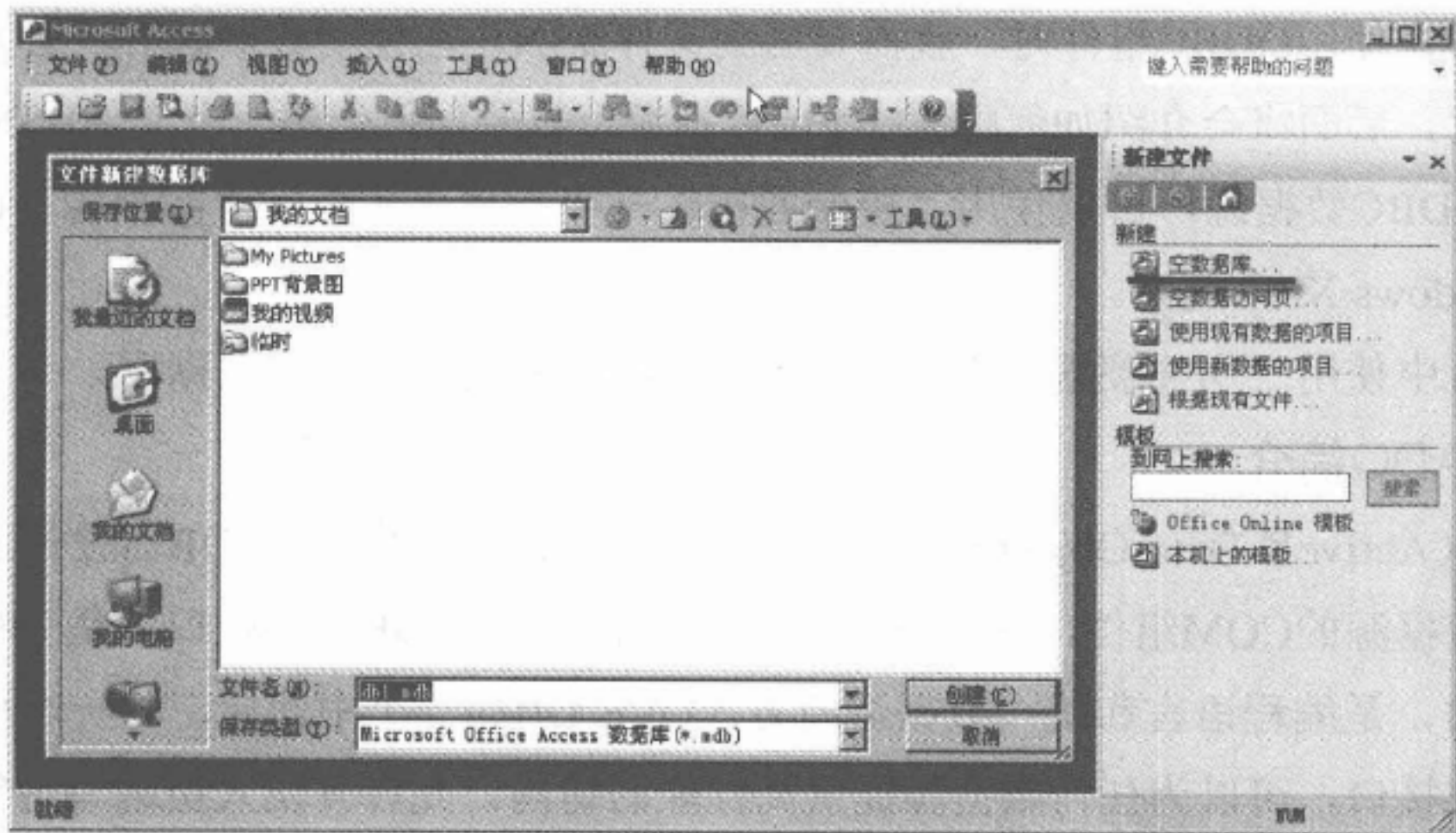


图6-1 新建Access数据库





第二步：弹出“文件新建数据库”对话框，指定新数据库的储存位置、文件名称（本例为“图书管理.mdb”），然后单击“创建”按钮。

第三步：创建完毕，出现名为“图书管理”的窗口，在“使用设计器创建表”上双击鼠标，如图6-2所示。

第四步：在“字段名称”输入表的第一个字段名称“书号”，在“数据类型”的下拉菜单中选“数字”。依此类推，分别设置“书名”、“作者”、“更换”、“数量”、“价格”、“出版社”、“入库日期”、“备注”等字段名称及该字段的数据类型，然后关闭表，此时，Office助手会询问您是否保存表，可选择“是”，如图6-3所示。

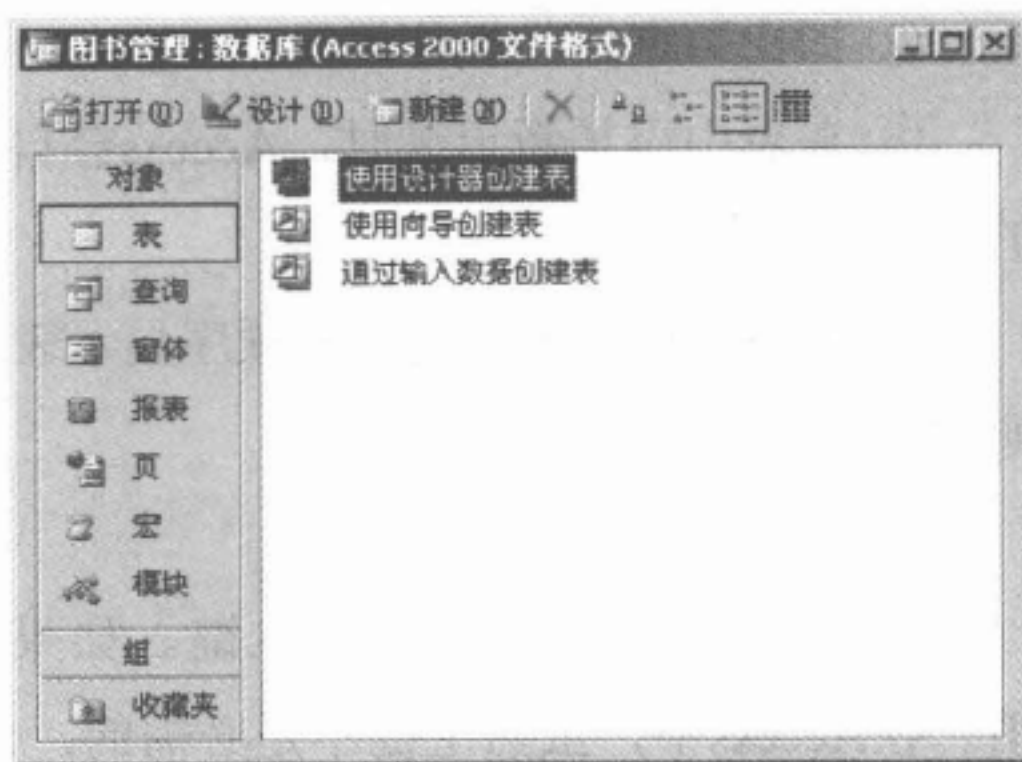


图6-2 创建Access数据表

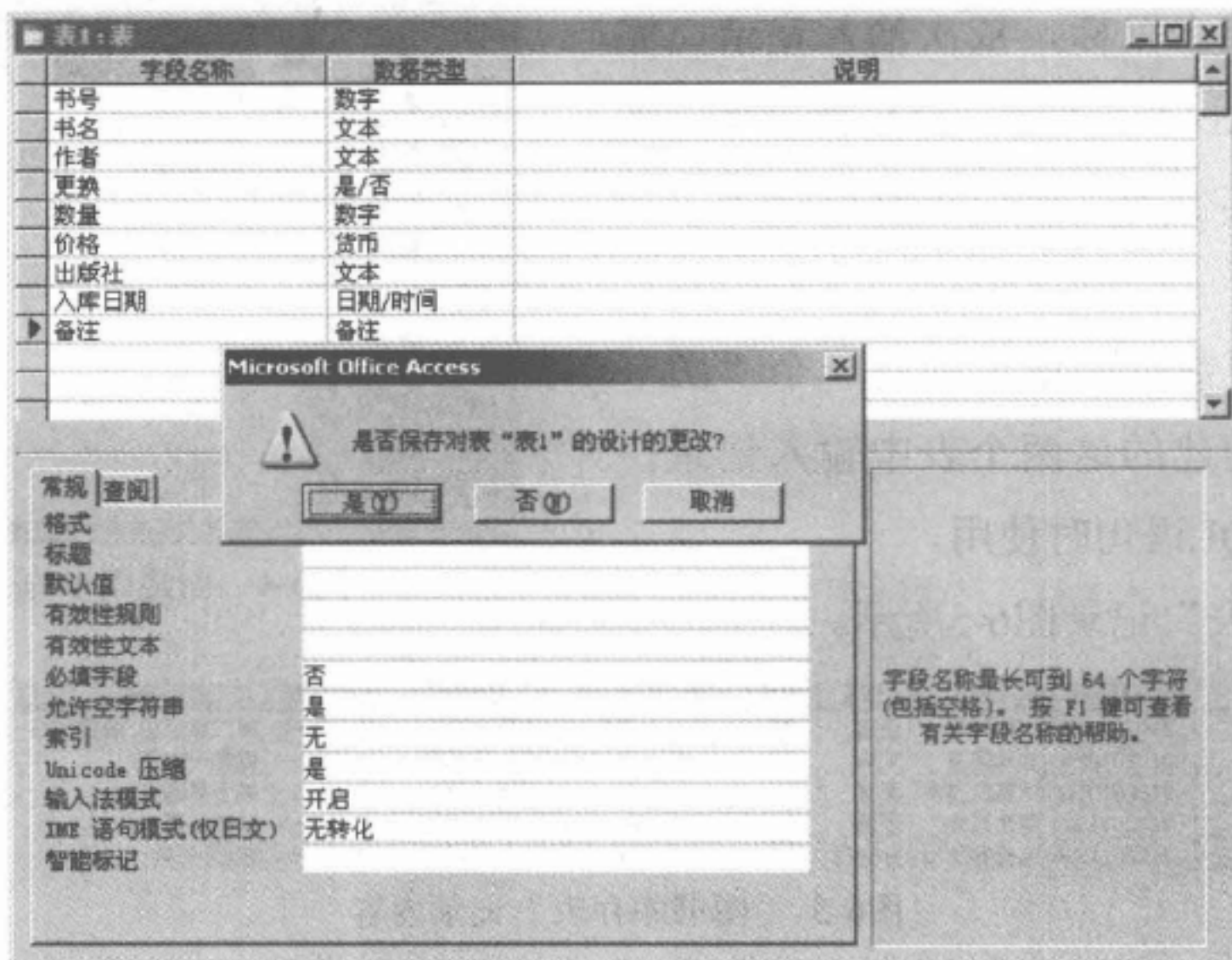


图6-3 输入Access数据表的字段

第五步：在“另存为”对话框的“表名称”框中输入表的名称为“图书库存表”，然后按“确定”。

第六步：在确认是否为数据表创建主键之后，就可以完成数据表的设计工作了。那么什么是主键呢？

主键是主关键字（primary key）的简称，是被挑选出来、作为表中记录行唯一标识的候选关键字。一个表只有一个主关键字。主键可以由一个字段，也可以由多个字段组成，分别称为单字段主键或多字段主键。

主键的作用主要有：

① 保证实体的完整性；





② 加快数据库的操作速度;

③ 在表中添加新记录时, Access会自动检查新记录的主键值, 不允许该值与其他记录的主键值重复;

④ Access自动按主键值的顺序显示表中的记录。如果没有定义主键, 则按输入记录的顺序显示表中的记录。

主键的特点主要有:

① 一个表中只能有一个主键。如果在其他字段上建立主键, 则原来的主键就会取消。在Access中, 虽然主键不是必需的, 但最好为每个表都设置一个主键;

② 主键的值不可重复, 也不可为空(NULL)。

完成了数据表的设计工作后可以在“图书管理”数据库窗口中看到新增的名为“图图书馆存表”的数据表, 如图6-4所示。

第七步: 在“图书管理”数据库窗口新增表的名称上双击鼠标, 依次输入记录, 完毕后可以关闭Access, Access会自动存盘。如果需要再建立其他表, 可以重复以上第三步~第七步。

在图书管理数据库中再创建一个“图书租借表”, 在创建的这两个表中输入记录, 以备后面介绍SQL语句时使用。

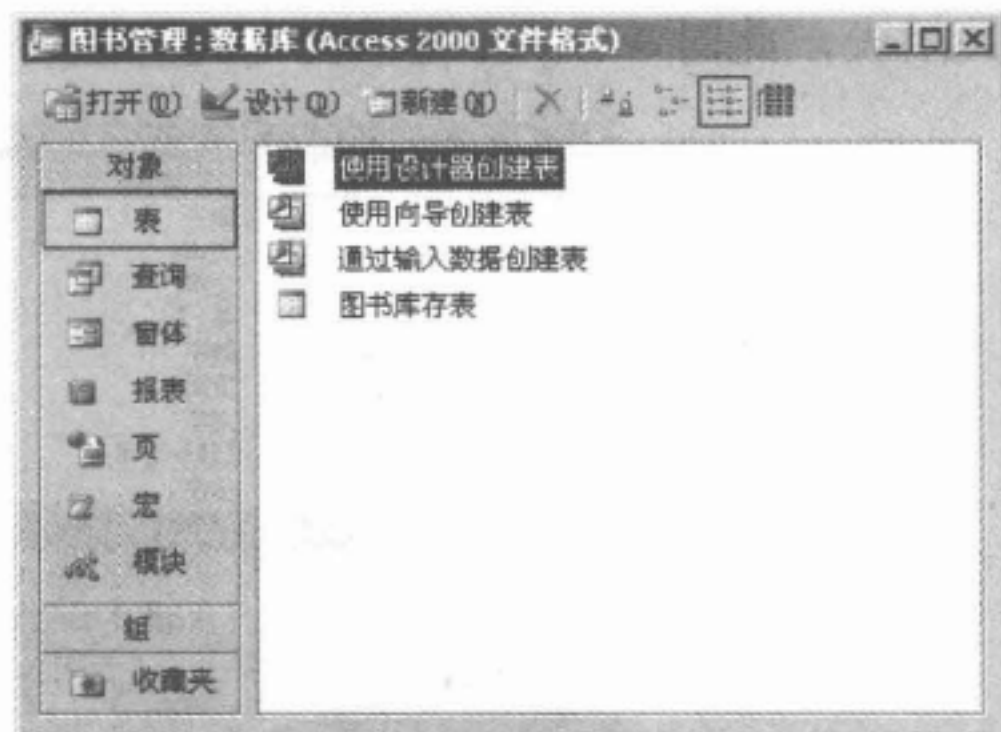


图6-4 创建图图书馆存表成功

“图图书馆存表”记录图6-5所示。

书号	书名	作者	更换	数量	价格	出版社	入库日期	备注
2102010023	水稻种植	王道	<input checked="" type="checkbox"/>	10	¥ 59.00	大连出版社	1998-10-16	第二版已出, 需要更换
2102010145	幼儿教育	刘辉	<input type="checkbox"/>	5	¥ 32.00	广州出版社	1990-12-1	就剩一本了
2102010510	计算机技术	彭章	<input type="checkbox"/>	8	¥ 147.00	阳光出版社	2000-5-3	关于易语言的
2102011049	湖南风光	于渊	<input type="checkbox"/>	20	¥ 231.00	人民出版社	2005-4-26	缺第78页
2102011596	流感的防治	孙家行	<input type="checkbox"/>	12	¥ 65.00	北京出版社	2001-9-16	在6号书架

图6-5 “图图书馆存表”记录内容

“图书租借表”记录如图6-6所示。

书号	书名	借书人	借书时间	还书时间
2102010510	计算机技术	邓冷	2008-3-16	2008-4-1
2102011049	湖南风光	冯可	2009-5-1	2009-5-26
2102010145	幼儿教育	冯可	2010-1-11	2010-2-2

图6-6 “图书租借表”记录内容

## 6.2 SQL语言应用

SQL (Structured Query Language, 结构化查询语言) 语言是一种数据库查询和程序设计语言, 是对关系型数据库进行访问的标准语言, 用于存取数据以及查询、更新和管理关





系数据库系统。例如：Access、SQL Server、Mysql等数据库都支持SQL语言。所以学习操作外部数据库前，建议先对SQL语言作一些了解。

SQL语言具有功能丰富、使用方便灵活、语言简洁等优点。1986年10月，美国国家标准局（ANSI）批准了SQL作为关系数据库语言的美国国家标准，此后不久，国际化标准组织（ISO）也做出了同样的决定，使SQL成为了国际标准。

SQL语言是高级的非过程化编程语言，允许用户在高层数据结构上工作。它不要求用户指定对数据的存放命令，也不需要用户了解具体的数据存放方式，所以具有完全不同底层结构的不同数据库系统可以使用相同的SQL语言作为数据输入与管理的接口。它以记录集合作为操作对象，所有SQL语句以接受集合作为输入，以返回集合作为输出，这种集合特性允许一条SQL语句的输出作为另一条SQL语句的输入，所以SQL语句可以嵌套，这使它具有极大的灵活性和强大的功能，在多数情况下，在其他语言中需要一大段程序实现的功能只需要一个SQL语句就可以达到目的。

SQL语言包括三种主要程序设计语言类别：数据定义语言（DDL）、数据操作语言（DML）及数据控制语言（DCL）。SQL语言的每个功能都由若干条指令组成，每条指令表示对数据库的一种操作。

## 6.2.1 常用的SQL语句

常用的SQL语句包括：

DML（Data Manipulation Language，数据操作语言）：用于检索或者修改数据；

SELECT：用于检索数据；

INSERT：用于增加数据到数据库；

UPDATE：用于从数据库中修改现存的数据；

DELETE：用于从数据库中删除数据；

DDL（Data Definition Language，数据定义语言）：用于定义数据的结构（如：创建、修改或者删除数据库对象）；

CREATE TABLE：用于创建表；

ALTER TABLE：用于修改表；

DROP TABLE：用于删除表；

CREATE INDEX：用于创建索引；

DROP INDEX：用于删除索引；

DCL（Data Control Language，数据控制语言）：用于定义数据库用户的权限；

GRANT：用于赋予一个用户，一个组或所有用户访问权限；

REVOKE：用于废除某用户或某组或所有用户访问权限。

常用的SQL语句就介绍这些，如果还要使用更高级的SQL语句，可以查看相关书籍和资料。





## 6.2.2 定义数据库用户的权限

支持用户管理的数据库系统可以定义数据库用户的权限。

### 1) 设置访问权限

SQL语言中的GRANT语句被用来设置用户对数据库的访问权限。GRANT语句的使用格式如下：

GRANT 权限 on 数据库对象 to 用户名

权限：所有权限（ALL），或单独的查询（SELECT）更新（UPDATE）等权限

数据库对象：赋予权限的对象名称，如表名，视图名，索引名等

用户名：可以是所有用户（public），或某个单独用户

在GRANT后面加入所需要的权限，然后再依次设定数据库对象和用户名等。

例如：允许所有用户查询图书库存表的内容。使用SQL语句：

```
GRANT select on 图书库存表 to public
```

在易语言中，使用“外部数据库”的代码：

```
外部数据库.执行 (“GRANT select on 图书库存表 to public”,)
```

### 2) 删除访问权限

SQL语言中的REVOKE语句被用来废除用户对数据库的访问权限。REVOKE语句的使用格式如下：

REVOKE 权限 on 数据库对象 from 用户名

权限：所有权限（ALL），或查询（SELECT）更新（UPDATE）等权限

数据库对象：赋予权限的对象名称，如表名，视图名，索引名等

用户名：可以是所有用户（public），或某个用户名

在REVOKE后面加入所需要的权限，然后再依次设定数据库对象和用户名等。

例如：禁止用户“陈名”删除图书库存表的内容。使用SQL语句：

```
REVOKE delete on 图书库存表 from 陈名
```

在易语言中，使用“外部数据库”的代码：

```
外部数据库.执行 (“REVOKE delete on 图书库存表 from 陈名”,)
```

## 6.2.3 定义表的结构

### 1) 创建表

SQL语言中的CREATE TABLE语句被用来建立新的数据表。CREATE TABLE语句的使用格式如下：

CREATE TABLE 表名

（第1个字段名 第1个字段数据类型 [字段长度]，





第2个字段名 第2个字段数据类型 [字段长度],  
第3个字段名 第3个字段数据类型 [字段长度]  
.....)

在CREATE TABLE后面加入所要建立的表的名称,然后在括号内顺次设定各字段的名称,数据类型等。

例如:要使用SQL语句创建一个名为“图书库存表”的数据表,有三个字段,第一个字段为“书号”,整数型;第二个字段为“书名”,文本型,文本长度是10;第三个字段为“数量”,整数型。使用SQL语句:

```
CREATE TABLE 图书库存表(书号 int,书名 text(10),数量 int)
```

在易语言中,使用“外部数据库”的代码:

```
外部数据库.执行(“CREATE TABLE 图书库存表(书号 int,书名 text(10),数量 int)” ,)
```

## 2) 删除表

SQL语言中的DROP TABLE语句被用来删除数据表。DROP TABLE语句的使用格式如下:

**DROP TABLE 表名**

在DROP TABLE后面加入所要删除的表的名称。

例如:要使用SQL语句删除一个名为“图书库存表”的数据表。使用SQL语句:

```
DROP TABLE 图书库存表
```

在易语言中,使用“外部数据库”的代码:

```
外部数据库.执行(“DROP TABLE 图书库存表” ,)
```

后面介绍的SQL语句,在使用“外部数据库”组件的执行命令时,都以此为例。

## 3) 修改表

SQL语言中的ALTER TABLE语句被用来修改数据表。ALTER TABLE语句的使用格式如下:

**ALTER TABLE 表名 子修改内容**

子修改内容有多种形式,在这就列举几种:

**ADD COLUMN:** 增加字段

**RENAME COLUMN** 旧字段名 **TO** 新字段名: 修改字段名

**RENAME TO** 新表名: 修改表名

在ALTER TABLE后面加入所要修改的表的名称,然后设置修改的内容。

例如:要使用SQL语句向“图书库存表”中加入一个字段,字段名称“作者”,字段类型为文本型,文本长度是10。使用SQL语句:

```
ALTER TABLE 图书库存表 ADD COLUMN 作者 text(10)
```

在易语言中,使用“外部数据库”的代码:

```
外部数据库.执行(“ALTER TABLE 图书库存表 ADD COLUMN 作者 text(10)” ,)
```







## 6.2.4 数据检索

### 1) 插入记录

要在表中插入记录使用INSERT语句，格式如下：

INSERT [INTO] 表名

(第1个添加记录的字段名，……最后1个添加记录的字段名)

[VALUES] (第1个添加的数据，……最后1个添加的数据)；

向表中插入数据，要在关键字INSERT INTO 之后紧跟着表名，在一对括号中，是以逗号分开的一系列的字段名；然后在关键字VALUES之后跟着一系列用括号括起的数据。这些数据必须与指定的字段名相匹配。

例如，要在图书库存表中插入一个记录，

```
insert into 图书库存表(书号,书名,作者,更换,数量,价格,出版社,入库日期,备注)
values (3003,' 易语言教程' , ' 吴涛' ,1,10,88,' 易语言出版社' , ' 2010-1-19' ,
' 内容备注' )
```

在易语言中，使用“外部数据库”的代码：

```
外部数据库.执行(“insert into 图书库存表(书号,书名,作者,更换,数量,价格,
出版社,入库日期,备注) values (3003,' 易语言教程' , ' 吴涛' ,1,10,88,'
易语言出版社' , ' 2010-1-19' ,?)” ,备注变量)
```

备注型字段和字节集型字段可以这样插入记录：用“？”代替要插入的内容，在第二个参数中设置要插入的内容变量，具体请参见后面的“外部数据库.执行”命令的用法。

#### 注解：

- (1) SQL语句中关键字的前后要有空格。
- (2) 多个表名、多个字段名或多个数据之间要加上逗号。
- (3) 文本型、日期型数据要用单引号引上。
- (4) 字节集型数据的使用请参看后面“外部数据库.执行”的用法。

### 2) 删除记录

要从表中删除一个或多个记录，需要使用DELETE语句。给DELETE 语句提供WHERE 子句。WHERE子句用来选择要删除的记录。格式如下：

DELETE [FROM 表名]

[WHERE 条件语句]

FROM语句后是欲删除记录的表名，WHERE语句中填写要删除记录所符合的条件。

**注解：**在没有指定WHERE子句时，DELETE语句将删除表中的所有记录，所以使用的时候要十分注意。





例如：删除图书库存表中，书名为“易语言教育”的记录，使用SQL语句如下：

```
DELETE FROM 图书库存表 WHERE 书名='易语言教育'
```

3) 修改记录

要修改表中已经存在的一条或多条记录，应使用 UPDATE语句。UPDATE语句格式如下：

```
UPDATE 表名
SET 字段及内容[字段及内容,...]
[WHERE 条件表达式]
```

在UPDATE后面是欲操作的表名，SET后面是欲改变的字段及内容，然后在WHERE子句中填写被改变记录要符合的条件。

例如：前面图书库存表中，将书籍“计算机技术”的数量改为8，使用SQL语句如下：

```
UPDATE 图书库存表 SET 数量=8 WHERE 书名='计算机技术'
```

当然，UPDATE语句也可以修改多个表、多个字段的内容，比如：要将图书库存表和图书租借表中书号为2102011049的书名改为“易语言”。使用以下SQL语句：

```
UPDATE 图书库存表,图书租借表 SET 图书库存表.书名='易语言',图书租借表.书名='易语言' WHERE 图书库存表.书号=2102011049 and 图书租借表.书号=2102011049
```

4) 查询语句SELECT

SELECT语句是很常用的SQL语句，语法也很简单，但是通过变化查询条件和查询方式，SELECT语句可以完成多种查询任务，格式如下：

```
SELECT 目标字段
FROM 基本表
[WHERE 条件表达式]
[GROUP BY 字段名1] 将数据相同的行集中在一起，允许合计函数计算
[HAVING内部函数表达式] 设定记录的条件，应该处在GROUP BY子句之后
[ORDER BY 字段名2 ASC/DESC] 根据指定的字段排序，默认升序（ASC）
在WHERE子句中可以有以下的条件选择： WHERE子句中常用运算符及意义。
SQL语句运算符见表6-1。
```

表6-1 SQL语句运算符

运算符	意 义	运算符	意 义
=	等于	LIKE	类似于（支持通配符）
>	大于	NOT LIKE	不类似于（支持通配符）
<	小于	AND	AND连接的多个条件必须同时满足
>=	大于等于	OR	OR连接的多个条件至少满足其中1个
<=	小于等于	NOT	排除后跟条件
<>	不等于		





### (1) 简单查询

例如：查询“图书库存表”中易语言出版社出版的书籍，使用SQL语句：

```
SELECT * FROM 图书库存表 WHERE 出版社=' 易语言出版社'
```

语句中的“\*”代表查询和返回所有字段，FROM后面的“图书库存表”是所查询的表的名称。WHERE后面的表达式，是要查询的条件。

在易语言中，如果使用“外部数据库”组件，可以用下面代码执行上面的SQL语句：

```
外部数据库.查询 (“SELECT * FROM 图书库存表 WHERE 出版社=' 易语言出版社'”)
```

后面介绍的SQL语句，在使用“外部数据库”组件查询时，都以此为例。

例如：查询“图书库存表”中价格小于90的所有书籍，并返回“书名”和“作者”的记录内容，最后按价格的升序排列查找到的结果。使用SQL语句：

```
SELECT 书名,作者 FROM 图书库存表 WHERE 价格<90 ORDER BY 价格 ASC
```

查询的结果将只返回符合条件的记录的书名字段和作者字段的内容，由于有ORDER BY子句：“ORDER BY 价格 ASC”，所以返回的记录将会按价格的升序排列。

例如：查询“图书库存表”中数量在10至20之间的书籍。使用SQL语句：

```
SELECT * FROM 图书库存表 WHERE 数量 BETWEEN 10 AND 20
```

例如：查询“图书库存表”中书名开头包含“易”字的记录，使用SQL语句如下：

```
SELECT * FROM 图书库存表 WHERE 书名 LIKE ' 易%'
```

例程中使用了LIKE语句，LIKE语句的格式是：

字段名 LIKE 指定字符串

其中的字符串有以下几种形式：

“%”：任意长度的任意字符。“\_”：下划线代表任意单个字符。其他字符：仅代表自身。所以上面的SQL语句就查询了所有书名中第一个字是“易”的记录。

### (2) 联合查询

如果查询中涉及了两个以上的表，则称之为联合查询。在WHERE子句中引导的查询条件中字段的表示命令有些类似易语言中组件属性的表示命令，表示为：

查询表. 字段名

例如：查询两个表中被“冯可”租借出去的所有书籍的资料。使用SQL语句：

```
SELECT * FROM 图书库存表,图书租借表 WHERE 图书租借表.书号=图书库存表.书号  
and 图书租借表.借书人=' 冯可'
```

WHERE子句中的条件，两个字段进行比较，那这两个字段必须具有可比性，但不必相同，也可以用其他运算符号连接。

例如：查询两个表中价格小于60的书名、借书人。使用SQL语句：

```
SELECT 图书库存表.书名,图书租借表.借书人 FROM 图书库存表,图书租借表 WHERE  
图书租借表.书名=图书库存表.书名 and 图书库存表.价格<60
```





(3) 统计查询

想对一个表中的记录进行数据统计，需要用到统计函数，SQL统计函数见表6-2。

表6-2 SQL统计函数

函数名	意 义	函数名	意 义
MIN	返回一个给定列中最小的数值	AVG	返回一个给定列中所有数值的平均值
MAX	返回一个给定列中最大的数值	COUNT	返回一个给定列中所有数值的个数
SUM	返回一个给定列中所有数值的总和	COUNT(*)	返回一个表中的行数

① 记录数目

要统计图书库存表中，数量在10本以上的有几种书籍，可以使用以下SQL语句：

```
SELECT COUNT(数量) FROM 图书库存表 WHERE 数量>=10
```

② 取平均值

要统计图书库存表中，所有书籍价格的平均值，可以使用SQL语句：

```
SELECT AVG (价格) FROM 图书库存表
```

③ 求和

统计易语言出版社出版的图书的价格总和，可以使用SQL语句：

```
SELECT SUM (价格) FROM 图书库存表 where 出版社=' 易语言出版社'
```

④ 取最大值、取最小值

要取出图书库存表中所有书籍的最高价格，使用SQL语句：

```
SELECT MAX (价格) FROM 图书库存表
```

6.3 外部数据库组件

易语言中ODBC连接外部数据库的组件包括“外部数据库”和“外部数据提供者”，下面将主要以Access数据库为例介绍以ODBC方式操作外部数据库。

6.3.1 “外部数据提供者” 组件

外部数据提供者组件除基本属性外，还有两个重要的属性：“连接文本”和“查询SQL”。

外部数据提供者组件主要通过数据源对外部数据进行操作。

“连接文本”属性：文本型，用于设置外部数据库的ODBC连接文本。

“查询SQL”属性：文本型，用作指定数据库中的数据表名或者用作查询记录集的SELECT类SQL语句。

下面来看看外部数据提供者是如何通过数据源连接Access数据库的。





第一步：新建一个易程序，在窗口界面右边的组件箱中选择“外部数据提供者”组件，将其拖放到“启动窗口”上，在窗口界面左边的属性面板中选中“连接文本”属性，点击出现的小按钮，如图6-7所示。

**注解：**“外部数据提供者”组件是不可视组件，不需要设置它的位置和大小。

第二步：在弹出的“选择数据源”对话框中选择数据源，如图6-8所示。

第三步：单击“新建”按钮，弹出一个“创建新数据源”对话框。选中“Driver do Microsoft Access”（Access的驱动程序），然后单击“下一步”按钮，如图6-9所示。

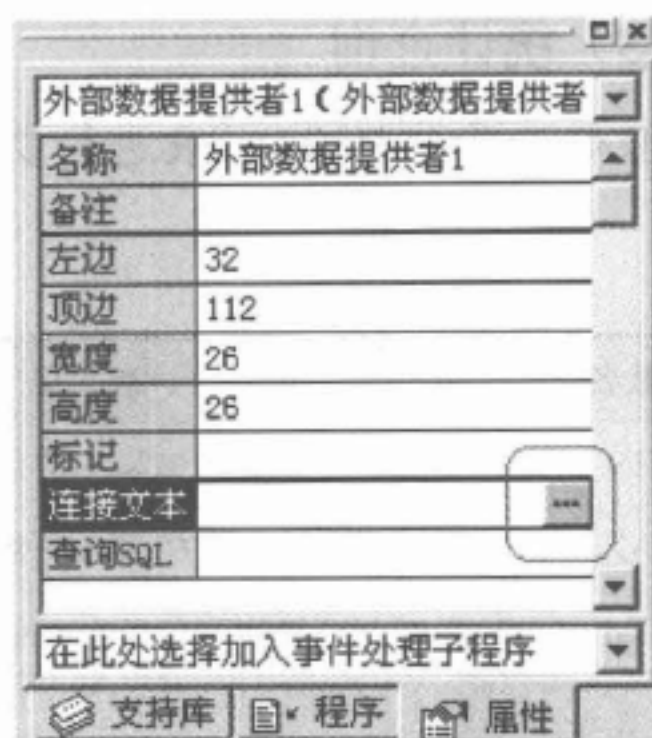


图6-7 “外部数据提供者”的连接文本属性

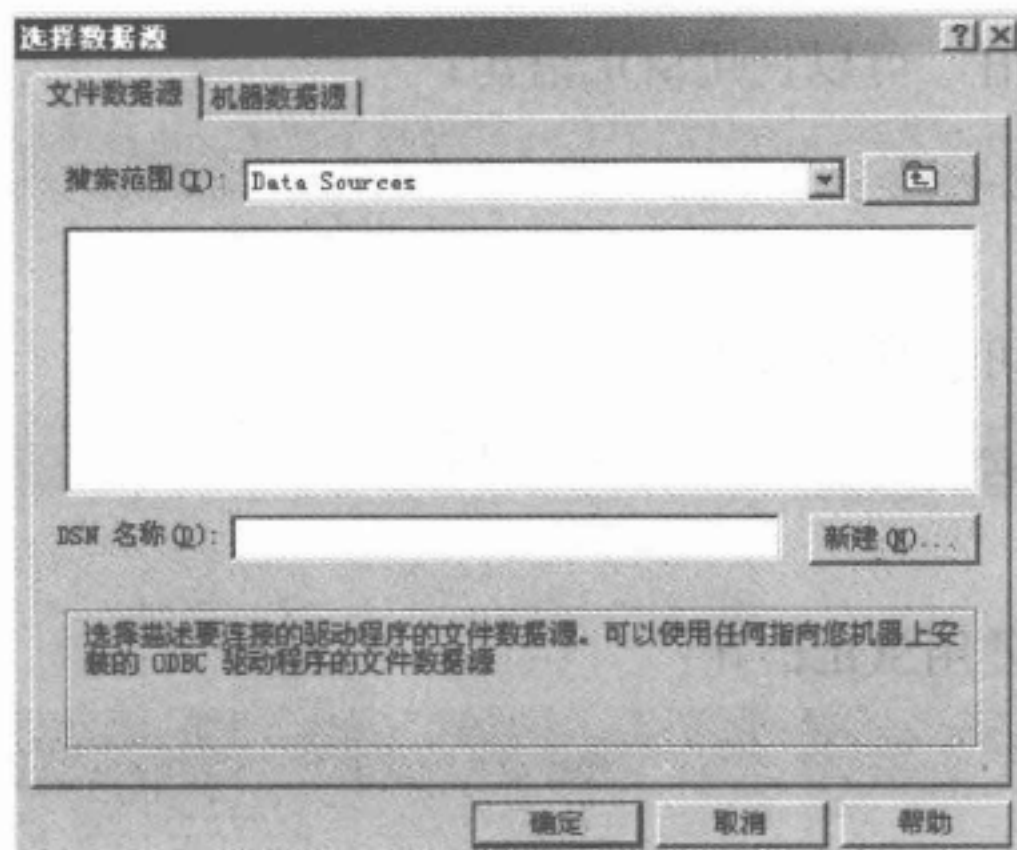


图6-8 选择数据源



图6-9 选择数据源驱动程序

第四步：单击“浏览”按钮。在弹出的对话框中选择路径，输入“图书管理.dsn”，点击“下一步”按钮，如图6-10所示。

第五步：单击“下一步”按钮后，会弹出确认对话框，列出前几步所设置的信息。如果确认无误，点击“完成”，创建数据源文件工作到此结束，如图6-11所示。



图6-10 数据源文件名



图6-11 创建数据源文件





第六步：建立“图书管理.dsn”数据源文件（已设置了路径和驱动程序信息，但还没有和具体的数据库相连接）后，单击“完成”按钮，会弹出一个对话框。可在这个对话框中通过单击“选择”按钮连接数据库，如图6-12所示。

第七步：在弹出的“选择数据库”对话框中选择书中所提供的Access数据库文件“图书管理.mdb”，然后单击“确定”按钮，如图6-13所示。



图6-12 连接数据库



图6-13 选择数据库

第八步：可以发现，弹出的对话框与第六步选择数据库时是一个对话框，只是中间多了一行数据库的路径名，这正是所要连接的数据库（见图6-14）。在选择完数据库后，单击“确定”按钮，还会弹出“选择数据源”对话框。

第九步：可以发现，弹出的对话框与第二步选择数据库时是一个对话框，在“搜索范围”中选择刚才保存的“图书管理.dsn”数据源文件，单击“确定”按钮，就完成了数据源的连接，如图6-15所示。

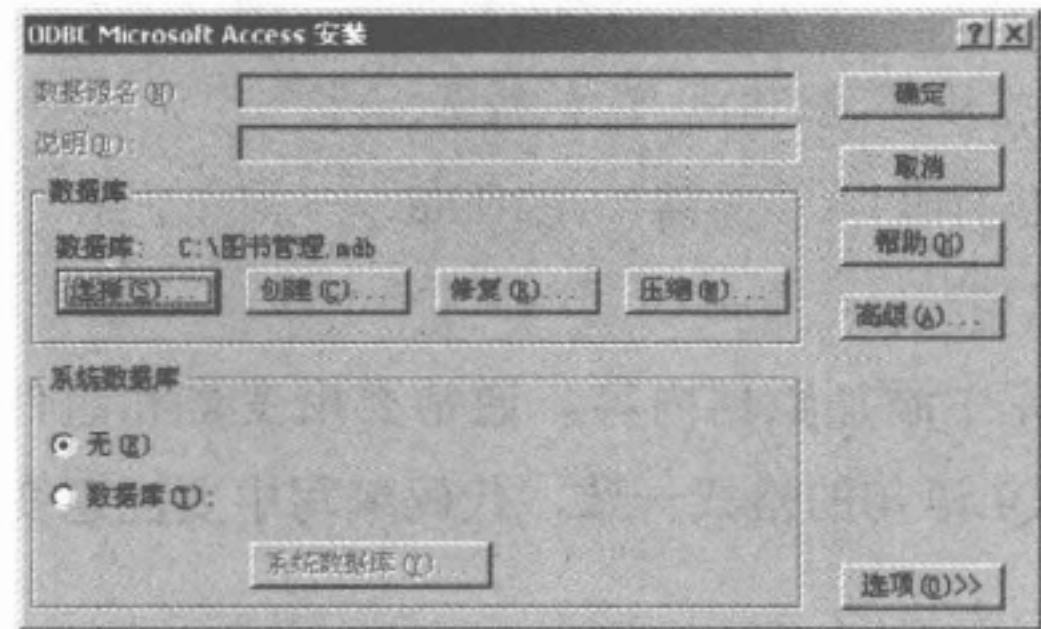


图6-14 连接数据源文件

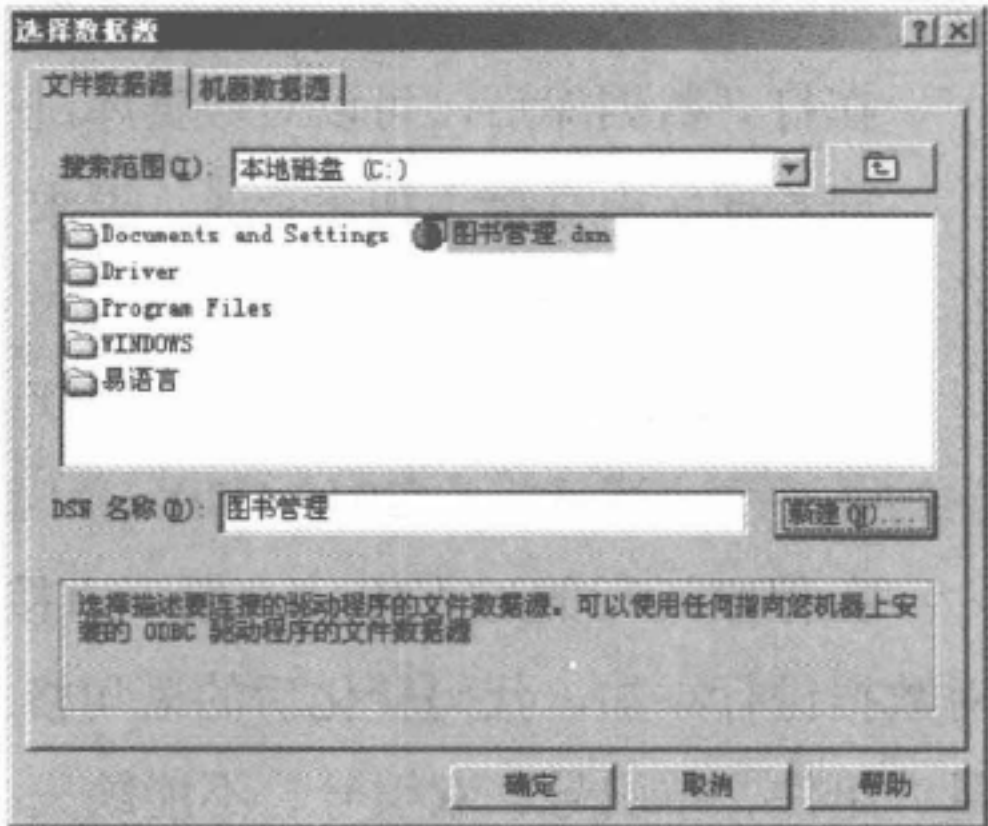


图6-15 完成ODBC的连接

此时，会出现与第八步一样的界面（见图6-14），单击“确定”按钮，就完成全部操作了。

至此，ODBC配置的标准操作过程全部完成。当下次再连接同一数据库时，就不必再新建一个dsn数据源文件了，直接在“选择数据源”对话框中选择“图书管理.dsn”就行了。





## 6.3.2 应用实例

连接上数据库后,就可以对数据库进行操作了。先来看一下外部数据库组件对Access数据库基本操作。

第一步:新建一个易语言程序,加入组件,设计界面,如图6-16所示。

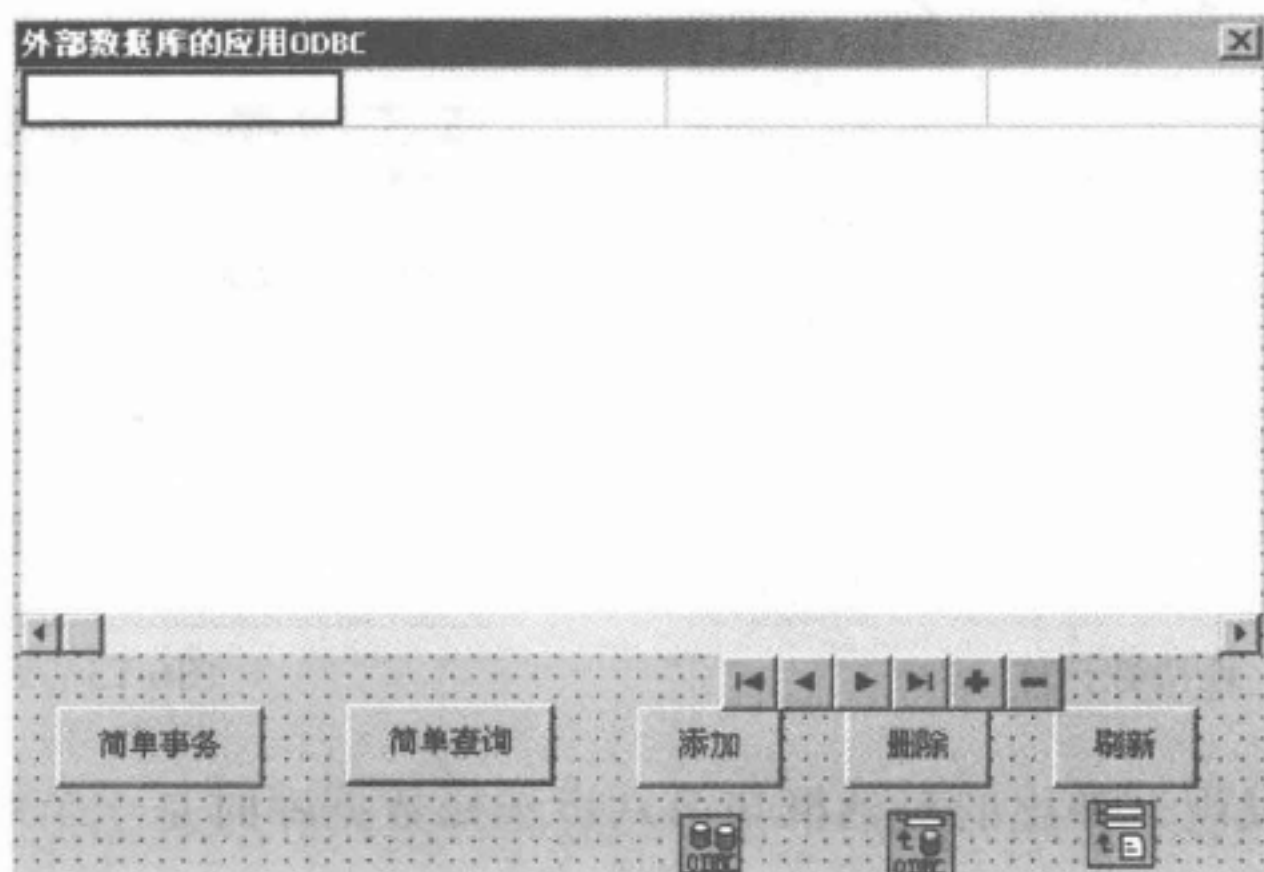


图6-16 应用外部数据库组件的界面设计

加入的组件有:表格、数据源、按钮、外部数据库、外部数据提供者、通用提供者。接下来编写代码,完成数据库的基本操作功能。

第二步:应用“外部数据库.打开MDB数据库( )”命令打开Access数据库。

外部数据库.打开MDB数据库(取运行目录() + “\图书管理.mdb”,,,)

在打开数据库后,可以对数据库进行添加、删除、修改、查询等操作。

向数据库中直接添加记录时,要求字段名与数据一一对应,数据类型要准确。

外部数据库.执行(“insert into 图书库存表(书号,书名,作者,更换,数量,价格,出版社,入库日期,备注) values (3003, '易语言教程', '吴涛', 1, 10, 88, '易语言出版社', '2010-1-19', '备注内容')”,)

在程序运行的过程中,一般不直接向数据库中添加数据内容,通常要用变量和组件对数据进行添加,代码转化后的语句格式要与SQL语句的格式一致,代码编写中要注意变量、组件、单引号、双引号、不能缺失。

外部数据库.执行(“insert into 图书库存表(书号,书名,作者,更换,数量,价格,出版社,入库日期,备注) values (“ + 书号编辑框.内容 + “,” + 书名编辑框.内容 + “,” + 作者编辑框.内容 + “,” + 到文本(选择(更换选择框.选中, 1, 0)) + “,” + 数量编辑框.内容 + “,” + 价格编辑框.内容 + “,” + 出版社编辑框.内容 + “,” + “,” + 日期格式 + “,” + 备注编辑框.内容 + “”)”,)





第三步：删除图书库存表中书号是1号的记录。

外部数据库.执行 (“delete from 图书库存表 where 书号=1”,)

第四步：修改图书库存表中书名是“易语言教育”的记录的“数量”为6。

外部数据库.执行 (“update 图书库存表 set 数量=6 where 书名='易语言教育'”,)

第五步：使用“外部数据库.查询”命令查询表，得到包含数据表中全部记录的记录集句柄。

记录集句柄 = 外部数据库.查询 (“select \* from 图书库存表 order by 书号”)

还可以根据条件查询数据库的内容，得到的数据表里的记录都符合查询的条件。

记录集句柄 = 外部数据库.查询 (“select \* from 图书库存表 where 价格>60 and 出版社='易语言出版社'”)

根据查询得到的记录集句柄，使用循环判断，将数据表中的记录一一读出，并写到数据源中，因数据源与表格相连，因此会在表格中显示出数据表中的所有的记录。最后关闭记录集。

变量名	类型	静态	数组	备注
计次变量	整数型			

外部数据库.到首记录 (记录集句柄)

计次变量 = 1

--> 循环判断首 0

数据源1.添加行 0

计次变量 = 计次变量 + 1

数据源1.置文本 (计次变量, 1, 到文本 (计次变量 - 1))

数据源1.置文本 (计次变量, 2, 到文本 (外部数据库.读 (记录集句柄, 1)))

数据源1.置文本 (计次变量, 3, 到文本 (外部数据库.读 (记录集句柄, 2)))

数据源1.置文本 (计次变量, 4, 到文本 (外部数据库.读 (记录集句柄, 3)))

数据源1.置文本 (计次变量, 5, 到文本 (外部数据库.读 (记录集句柄, 4)))

数据源1.置文本 (计次变量, 6, 到文本 (外部数据库.读 (记录集句柄, 5)))

数据源1.置文本 (计次变量, 7, 到文本 (外部数据库.读 (记录集句柄, 6)))

数据源1.置文本 (计次变量, 8, 到文本 (外部数据库.读 (记录集句柄, 7)))

数据源1.置文本 (计次变量, 9, 到文本 (外部数据库.读 (记录集句柄, 8)))

数据源1.置文本 (计次变量, 10, 到文本 (外部数据库.读 (记录集句柄, 9)))

外部数据库.到后一记录 (记录集句柄)

--- 循环判断尾 (外部数据库.尾记录后 (记录集句柄) = 假)

外部数据库.关闭记录集 (记录集句柄)

外部数据库组件的全部命令的使用看下一节的内容。

6.3.3 “外部数据库” 组件

“外部数据库” 组件只有简单的“名称”、“备注”、“左边”、“顶边”、“宽度”、“高度”、“标记”等基本属性。它主要通过命令对外部数据库进行操作。





## 1) “打开 ()” 命令

<逻辑型> 对象. 打开 ([文本型 ODBC数据源连接文本], [逻辑型 是否只读], [逻辑型 不显示ODBC连接对话框])

打开指定的ODBC数据源以供以后操作。成功返回真，失败返回假。

参数<1>的名称为“ODBC数据源连接文本”，类型为“文本型(text)”，可以省略。指定指向ODBC数据源的连接文本，如果为空，则激活ODBC数据源选择对话框。如果参数被省略，默认值为空。

参数<2>的名称为“是否只读”，类型为“逻辑型(bool)”，可以省略。指定是否以只读方式打开。如果参数被省略，默认值为假。

参数<3>的名称为“不显示ODBC连接对话框”，类型为“逻辑型(bool)”，可以省略。指定当数据源连接失败时，是否自动显示ODBC的数据源连接对话框。如果参数被省略，默认值为假。

例程使用的连接文本：

```
"ODBC;DBQ=C:\图书管理.mdb;Driver={Driver do Microsoft Access (*.mdb)};  
DriverId=25;FIL=MSAccess;FILEDSN=C:\图书管理.dsn;MaxBufferSize=2048;  
MaxScanRows=8;PageTimeout=5;SafeTransactions=0;Threads=3;UID=admin;  
UserCommitSync=Yes;"
```

一般情况下，都是使用数据库的名称来打开数据库，很少要用到连接文本。在打开之前不确定是哪种数据库时要用数据源去连接。

使用连接文本打开Access数据库

外部数据库.打开(外部数据提供者.连接文本,,)

使用数据库文件名打开Access数据库

外部数据库.打开MDB数据库("C:\图书管理.mdb",,,)

## 2) “关闭 ()” 命令

<无返回值> 对象. 关闭 ()

关闭当前被打开的数据库。

例如：

外部数据库.打开(,,) ' 打开，在使用外部数据库之前调用

外部数据库.关闭() ' 关闭，不再使用外部数据库时调用

数据库打开了就一定要关闭，否则在转换使用其他数据库时容易发生错误。

## 3) “取连接文本 ()” 命令

<文本型> 对象. 取连接文本 ()

返回当前被打开数据库的ODBC数据源连接文本。失败返回空文本。





例如：在输出面板中显示ODBC数据源连接文本。

输出调试文本(外部数据库.取连接文本())

#### 4) “启动事务 ()” 命令

<逻辑型> 对象. 启动事务 ()

启动当前被打开数据库的事务，成功返回真，失败返回假。

数据库事务(transaction)就是一组SQL语句，这组SQL语句是一个逻辑工作单元。可以认为事务就是一组不可分割的SQL语句，其结果应该作为一个整体永久性地修改数据库的内容，或者作为一个整体取消对数据库的修改。

事务是这样一种机制，它确保多个SQL语句被当作单个工作单元来处理。同时进行的查询和更新彼此不会发生冲突，其他用户不会看到发生了变化但尚未提交的数据。一旦系统故障，数据库会自动地完全恢复未完成的事务。

进行事务控制的主要流程如下：

{ “启动事务”

.....

进行多项数据库操作，一旦有一项失败则“回滚事务”，使数据恢复到“启动事务”之前的状态。

.....

所有数据库操作全部成功完成，则“提交事务”。}

#### 5) “回滚事务 ()” 命令

<无返回值> 对象. 回滚事务 ()

回滚当前被打开数据库的事务，所有在上一次事务提交前对数据库所作的修改均被取消。

#### 6) “提交事务 ()” 命令

<无返回值> 对象. 提交事务 ()

提交当前被打开数据库的事务。

在程序中的体现。

子程序名	返回值类型	公开	备注
_事务按钮_被单击			

外部数据库.启动事务 ()

记录集句柄 = 外部数据库.查询 (“select \* from 图书库存表”)

中间对数据库一系列的操作

--- 如果真 (记录集句柄 = 0)

外部数据库.回滚事务 ()

返回 ()

外部数据库.提交事务 ()







## 7) “查询 ()” 命令

<整数型> 对象. 查询 (文本型 查询类SQL语句)

对当前被打开数据库进行数据查询，成功返回结果记录集句柄，失败返回0。

**注意：**当不再使用此记录集时，必须使用“关闭记录集”命令将其关闭。

记录集是查询数据库时符合查询条件的记录集合。记录集句柄是一个整数值，表示当前记录集的一个标志号码，类似于文件号，窗口句柄等。记录集句柄是该记录集的唯一标识。数据库被重新查询，记录集将发生改变，记录集句柄同样也改变。

参数<1>的名称为“查询类SQL语句”，类型为“文本型 (text)”。本参数提供用作查询的SELECT语句。

例如：查询图书库存表的所有记录，按书号排序（默认为升序ASC）。

记录集句柄 = 外部数据库.查询 (“select \* from 图书库存表 order by 书号”)

## 8) “重新查询 ()” 命令

<逻辑型> 对象. 重新查询 (整数型 记录集句柄)

对指定的记录集进行重新查询，并将其当前记录指针重定位到首记录。成功返回真，失败返回假。

参数<1>的名称为“记录集句柄”，类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

例如：对已查询的记录重新查询，以便更新记录数据。

外部数据库.重新查询 (记录集句柄)

**注解：**数据库被改动后，需要及时更新记录集。

## 9) “关闭记录集 ()” 命令

<无返回值> 对象. 关闭记录集 ([整数型 记录集句柄])

关闭指定的记录集。

参数<1>的名称为“记录集句柄”，类型为“整数型 (int)”，可以被省略。指定欲关闭记录集的句柄，记录集句柄由“查询”命令返回。如果参数值为0，则关闭所有记录集。如果参数被省略，默认值为0。

例如：关闭查询的记录集。

外部数据库.关闭记录集 (记录集句柄)

## 10) “首记录前 ()” 命令

<逻辑型> 对象. 首记录前 (整数型 记录集句柄)

如果指定记录集的当前记录指针已在首记录的前面，返回真，否则返回假。

参数<1>的名称为“记录集句柄”，类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。





## 11) “尾记录后 ()” 命令

<逻辑型> 对象. 尾记录后 (整数型 记录集句柄)

如果指定记录集的当前记录指针已在尾记录的后面, 返回真, 否则返回假。

参数<1>的名称为“记录集句柄”, 类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

在“外部数据库”组件中没有取记录集数量的命令, 所以在循环读取记录集记录时, 可以移动记录指针用“对象. 首记录前 ()”或“对象. 尾记录后 ()”来判断指针的位置。

## 12) “到首记录 ()” 命令

<无返回值> 对象. 到首记录 (整数型 记录集句柄)

将指定记录集的当前记录指针移动到第一条记录上。

参数<1>的名称为“记录集句柄”, 类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

## 13) “到尾记录 ()” 命令

<无返回值> 对象. 到尾记录 (整数型 记录集句柄)

将指定记录集的当前记录指针移动到最后一条记录上。

参数<1>的名称为“记录集句柄”, 类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

## 14) “到前一记录 ()” 命令

<无返回值> 对象. 到前一记录 (整数型 记录集句柄)

将指定记录集的当前记录指针向前移动一条记录。

参数<1>的名称为“记录集句柄”, 类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

## 15) “到后一记录 ()” 命令

<无返回值> 对象. 到后一记录 (整数型 记录集句柄)

将指定记录集的当前记录指针向后移动一条记录。

参数<1>的名称为“记录集句柄”, 类型为“整数型 (int)”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

“到首记录 ()”、“到尾记录 ()”、“到前一记录 ()”、“到后一记录 ()”命令的语法是相似的, 目的都是移动记录集的当前记录指针。

## 16) “读 ()” 命令

<通用型> 对象. 读 (整数型 记录集句柄, 通用型 字段名称或位置)

返回指定记录集的当前记录处指定字段的数据内容。如果因为找不到指定字段而导致读入失败将返回空文本, 如果因为其他原因而读入失败将返回对应该字段类型的空数据,







即：逻辑型返回假，日期时间型返回100年1月1日，文本型返回空文本，字节集型返回空字节集，其他所有数值型返回0。

**注解：**写入字段数据请用“UPDATE”SQL语句实现。

参数<1>的名称为“记录集句柄”，类型为“整数型（int）”。指定欲操作记录集的句柄。记录集句柄由“查询”命令返回。

参数<2>的名称为“字段名称或位置”，类型为“通用型（all）”。参数值可以为一个字段名称文本或者一个字段位置数值，字段位置数值从1开始。

所返回数据的类型与字段SQL类型对照情况见表6-3。

表6-3 返回值与SQL类型对应

返回数据类型	字段SQL类型
逻辑型	BIT
字节型	TINYINT
短整数型	SMALLINT
整数型	INTEGER
单精度型	REAL
双精度型	FLOAT、DOUBLE
日期时间型	DATE、TIME、TIMESTAMP
文本型	NUMERIC、DECIMAL、BIGINT、CHAR、VARCHAR、LONGVARCHAR
字节集型	BINARY、VARBINARY、LONGVARBINARY

在读取记录时，要注意指针的指向。通常先将当前记录指针移动到某记录上，然后再使用“读（）”的命令。返回值类型要与接收数据的变量类型一致。

17) “执行（）”命令

<逻辑型> 对象. 执行（文本型 非查询类SQL语句，[通用型 文本或字节集]，...）

执行指定的非查询类SQL语句，SQL语句中的列名（尤其是中文列名）可用中括号括起来。如果SQL语句中涉及到备注或者字节集型字段，请在相应位置加上问号，然后再加上对应的文本型或字节集型数据参数，参数数目必须与问号的数目一致。成功返回真，失败返回假。命令参数表中最后一个参数可以被重复添加。

参数<1>的名称为“非查询类SQL语句”，类型为“文本型（text）”。本参数提供非查询类SQL语句（无结果集返回），如修改、删除、添加等。

参数<2>的名称为“文本或字节集”，类型为“通用型（all）”，可以被省略。本参数仅用作为SQL语句中的备注或字节集型SQL参数（用问号标识其位置）提供相应数据，必须为文本或者字节集型（与SQL参数类型对应）。如果SQL语句中没有包含SQL参数，则应该省略本参数。



例如：向数据库中插入一个记录。

外部数据库.执行 (“insert into 图书库存表 (书号,书名,作者,更换,数量,价格,出版社,入库日期,备注) values (3003,' 易语言教程' , ' 吴涛' ,1,10,88,' 易语言出版社' , ' 2010-1-19' ,?)” ,备注内容变量)

#### 18) “打开MDB数据库 ()” 命令

<逻辑型> 对象. 打开MDB数据库 (文本型 MDB数据库文件名, [文本型 用户名], [文本型 密码], [逻辑型 是否只读], [逻辑型 不显示ODBC连接对话框])

本命令是针对 Access MDB 数据库的特定打开命令,通过自建 ODBC 连接文本来打开 MDB 数据库。调用本命令后无需再调用“打开”命令。成功返回真,失败返回假。

参数<1>的名称为“MDB数据库文件名”,类型为“文本型 (text)”。本参数提供即将打开数据库的文件名。

参数<2>的名称为“用户名”,类型为“文本型 (text)”,可以被省略。本参数提供打开数据库所需要的用户名,如果省略本参数,默认为不需要。

参数<3>的名称为“密码”,类型为“文本型 (text)”,可以被省略。本参数提供打开数据库所需要的密码,如果省略本参数,默认为不需要。

参数<4>的名称为“是否只读”,类型为“逻辑型 (bool)”,可以被省略。指定是否以只读方式打开。如果参数被省略,默认值为假。

参数<5>的名称为“不显示ODBC连接对话框”,类型为“逻辑型 (bool)”,可以被省略。指定当数据源连接失败时,是否自动显示ODBC的数据源连接对话框。如果参数被省略,默认值为假。

例如: 打开图书管理.MDB数据库。

外部数据库.打开MDB数据库 (取运行目录 () + “\图书管理.mdb” ,,,,)

#### 19) “打开SQL数据库 ()” 命令

<逻辑型> 对象. 打开SQL数据库 (文本型 服务器名称, [文本型 用户名], [文本型 密码], [文本型 数据库名称], [逻辑型 是否只读], [逻辑型 不显示ODBC连接对话框])

本命令是针对 SQL SERVER 数据库的特定打开命令,通过自建 ODBC 连接文本来打开 SQL SERVER 数据库。调用本命令后无需再调用“打开”命令。成功返回真,失败返回假。

参数<1>的名称为“服务器名称”,类型为“文本型 (text)”。本参数提供 SQL SERVER 服务器的名称或其客户端别名。

参数<2>的名称为“用户名”,类型为“文本型 (text)”,可以被省略。本参数提供登录服务器所需要的用户名,如果省略本参数,默认为不需要。





参数<3>的名称为“密码”，类型为“文本型(text)”，可以被省略。本参数提供登录服务器所需要的密码，如果省略本参数，默认为不需要。

参数<4>的名称为“数据库名称”，类型为“文本型(text)”，可以被省略。本参数提供即将打开数据库的名称。如果省略本参数，则使用该用户的默认数据库。

参数<5>的名称为“是否只读”，类型为“逻辑型(bool)”，可以被省略。指定是否以只读方式打开。如果参数被省略，默认值为假。

参数<6>的名称为“不显示ODBC连接对话框”，类型为“逻辑型(bool)”，可以被省略。指定当数据源连接失败时，是否自动显示ODBC的数据源连接对话框。如果参数被省略，默认值为假。

例如：使用IP地址打开SQL SERVER 数据库。

外部数据库.打开SQL数据库 (“192.168.0.1” ,,,,)

外部数据库组件的命令就简单介绍到这里。在使用外部数据库组件时，如果对帮助提示不能完全理解，可以参照本书中介绍的例如代码来学习外部数据库的应用。

## 6.4 数据库连接和记录集

易语言以ADO方式连接数据库的组件包括“数据库连接”组件和“记录集”组件。下面先简单介绍“数据库连接”组件和“记录集”组件对Access数据库的基本操作。

### 6.4.1 应用实例

新建一个易语言程序，加入组件，设计界面，如图6-17所示。

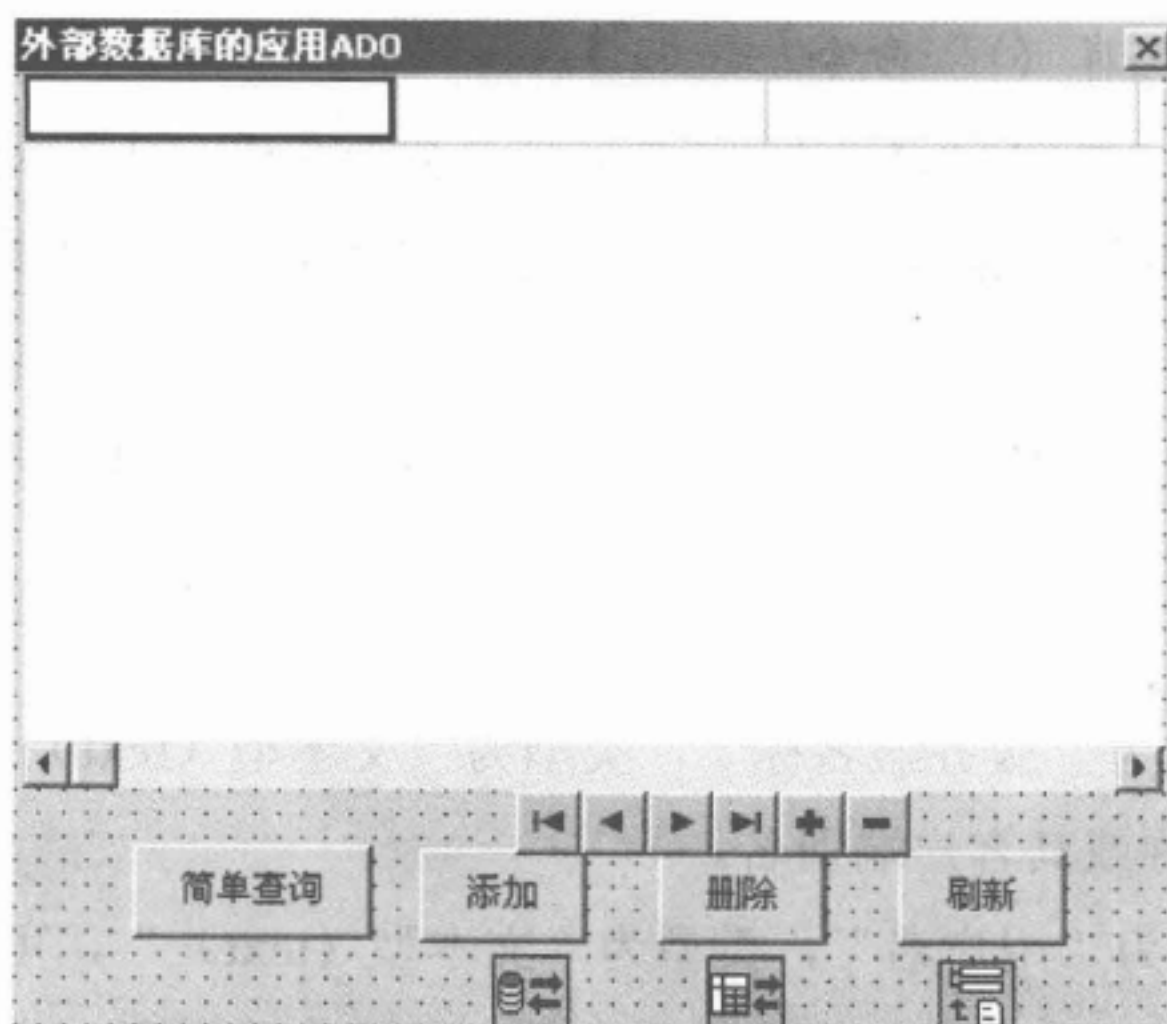


图6-17 应用外部数据库组件的界面设计





加入的组件有表格、数据源、按钮、数据库连接、记录集和通用提供者。接下来就是编写代码了。完成数据库操作的基本功能。

使用连接文本，应用“数据库连接.连接（）”打开数据库，或知道使用的数据库是“MDB”数据库时，用“数据库连接.连接Access（）”打开数据库；在打开数据库后，使用“记录集.置连接（）”连接到数据库连接组件，再使用“记录集.打开（）”命令打开表。

```
' 可以这样打开数据库
' 数据库连接.连接 ("Provider=Microsoft.jet.oledb.4.0;Data Source=图书管理.mdb;")
' 还可以这样打开数据库
数据库连接.连接Access (取运行目录 () + "\图书管理.mdb", "")
记录集.置连接 (数据库连接)
' 记录集.打开 ("图书库存表", #数据表名) ' 可以这样打开表
记录集.打开 ("select * from 图书库存表", #SQL语句) ' 还可以这样打开表
```

向数据库中添加记录时，先使用“记录集.添加”命令添加一条空记录，再向空记录中写入具体内容。最后更新数据库，完成记录添加。

记录集.添加 () ' 先添加再写入数据

```
记录集.写整数 ("书号", 到数值 (书号编辑框.内容))
记录集.写文本 ("书名", 书名编辑框.内容)
记录集.写文本 ("作者", 作者编辑框.内容)
记录集.写整数 ("数量", 到数值 (数量编辑框.内容))
记录集.写双精度 ("价格", 到数值 (价格编辑框.内容))
记录集.写文本 ("出版社", 出版社编辑框.内容)
记录集.写逻辑值 ("更换", 更换选择框.选中)
记录集.写日期时间 ("入库日期", 入库日期框.今天, 1)
记录集.写文本 ("备注", 备注编辑框.内容)
```

记录集.更新 () ' 添加完数据需要更新一下

还可以使用“数据库连接.执行SQL”命令直接向数据库中插入记录。

```
数据库连接.执行SQL ("insert into 图书库存表 (书号,书名,作者,更换,数量,价格,出版社,入库日期,备注) values (3003,' 易语言教程', ' 吴涛', 1,10,88,' 易语言出版社', ' 2010-1-19', ' 备注内容')")
```

删除记录集中第二条记录。注意记录号从0开始。

记录集.移到(1)

记录集.删除 (1)

还可以使用“数据库连接.执行SQL”命令直接从数据库中删除记录。

```
数据库连接.执行SQL ("delete from 图书库存表 where 书号=3003")
```

修改图书库存表中书名是易语言教育的记录的数量为5。





外部数据库.执行 (“update 图书库存表 set 数量=5 where 书名=’易语言教育’”,)

使用计次循环命令,将表中的记录一一读到对应的变量中,因数据源与表格相连,使用数据源将记录写到表格里。

**注:**“数据库连接.执行SQL”命令不能用来查询表的内容,因为“数据库连接.执行SQL”命令没有返回记录集的句柄,不能对所查询的记录集进行操作。

变量名	类型	静态	数组	备注
书号	整数型			
书名	文本型			
作者	文本型			
更换	逻辑型			
数量	整数型			
价格	双精度小数型			
出版社	文本型			
入库日期	日期时间型			
备注	文本型			
计次变量	整数型			

记录集.到首记录 ()

---> 计次循环首 (记录集.记录数量, 计次变量)

计次变量 = 计次变量 + 1

数据源1.添加行 ()

记录集.读整数 (“书号”, 书号)

记录集.读文本 (“书名”, 书名)

记录集.读文本 (“作者”, 作者)

记录集.读逻辑值 (“更换”, 更换)

记录集.读整数 (“数量”, 数量)

记录集.读双精度 (“价格”, 价格)

记录集.读文本 (“出版社”, 出版社)

记录集.读日期时间 (“入库日期”, 入库日期)

记录集.读文本 (“备注”, 备注)

数据源1.置文本 (计次变量, 1, 到文本 (计次变量 - 1))

数据源1.置文本 (计次变量, 2, 到文本 (书号))

数据源1.置文本 (计次变量, 3, 到文本 (书名))

数据源1.置文本 (计次变量, 4, 到文本 (作者))

数据源1.置文本 (计次变量, 5, 到文本 (更换))

数据源1.置文本 (计次变量, 6, 到文本 (数量))

数据源1.置文本 (计次变量, 7, 到文本 (价格))

数据源1.置文本 (计次变量, 8, 到文本 (出版社))

数据源1.置文本 (计次变量, 9, 到文本 (入库日期))

数据源1.置文本 (计次变量, 10, 到文本 (备注))

记录集.到下一条 ()

--- 计次循环尾 ()

数据库连接组件和记录集组件全部属性、命令的使用可看下面的内容。





## 6.4.2 “数据库连接” 组件

### 6.4.2.1 “数据库连接” 组件的属性

#### 1) “最后错误” 属性

只读，数据类型：文本型，获取最后错误文本。

#### 2) “是否已连接” 属性

只读，数据类型：逻辑型，判断数据库是否连接。

#### 3) “对象提供者” 属性

只读，数据类型：文本型，对象提供者名称。

#### 4) “引擎版本” 属性

只读，数据类型：文本型，数据引擎版本。

“数据库连接” 组件这些属性都是只读属性，即只能读取它的内容而不能更改。

### 6.4.2.2 “数据库连接” 组件的命令

#### 1) “连接 ()” 命令

<逻辑型> 对象. 连接 (文本型 连接文本)

连接到数据源，如果连接成功返回真，失败返回假。

参数<1>的名称为“连接文本”，类型为“文本型 (text)”。本参数提供连接时使用的连接文本。

例如：连接一个Access数据库代码如下：

```
数据库连接.连接 ("Provider=Microsoft.jet.oledb.4.0;Data Source=图书管理.mdb;")
```

连接一个SQL Server数据库代码如下：

```
数据库连接.连接 ("DRIVER=SQL Server;SERVER=192.168.0.1;UID=sa;PWD=123456;WSID=Administrator;DATABASE=图书管理")
```

连接文本中包括了连接数据库服务器、用户名、密码等信息。

如例句中的连接文本：“DRIVER”表示数据库服务器名、“SERVER”表示数据库的IP地址、“UID”表示用户名、“PWD”表示密码、“WSID”表示主机名、“DATABASE”表示操作的数据库名。

#### 2) “连接Access ()” 命令

<逻辑型> 对象. 连接Access (文本型 文件名, 文本型 密码)

连接Access数据库，如果连接成功返回真，失败返回假。

参数<1>的名称为“文件名”，类型为“文本型 (text)”。本参数提供Access数据库文件的完整路径名。

参数<2>的名称为“密码”，类型为“文本型 (text)”，初始值为“”。本参数提





供Access数据库的访问密码。可以为空，若为空，表示Access数据库没有密码。

例如：连接Access图书管理数据库。

```
数据库连接.连接Access("C:\图书管理.mdb", "")
```

### 3) “连接SQL Server ()” 命令

<逻辑型> 对象. 连接SQLServer (文本型 服务器名, 文本型 数据库名, 文本型 用户名, 文本型 密码)

连接SQL Server数据库，如果连接成功返回真，失败返回假。

参数<1>的名称为“服务器名”，类型为“文本型(text)”。本参数提供 SQL Server服务器名。

参数<2>的名称为“数据库名”，类型为“文本型(text)”。

参数<3>的名称为“用户名”，类型为“文本型(text)”。

参数<4>的名称为“密码”，类型为“文本型(text)”。

例如：连接SQL Server中的图书管理数据库。

```
数据库连接.连接SQL Server("192.168.0.1", "图书管理", "sa", "123456")
```

### 4) “关闭 ()” 命令

<逻辑型> 对象. 关闭 ()

断开当前数据库连接，成功返回真，失败返回假。

对数据库操作完毕时必须使用此命令来断开连接，否则与数据库的连接仍将持续一段时间，这样会造成服务器额外的负担。

例如：关闭当前数据库连接组件打开的数据库。

```
数据库连接.关闭 ()
```

### 5) “执行SQL ()” 命令

<逻辑型> 对象. 执行SQL (文本型 SQL语句)

执行指定的查询、SQL 语句、存储过程等，执行成功返回真，否则返回假。

参数<1>的名称为“SQL语句”，类型为“文本型(text)”。本参数提供要执行的SQL语句。

例如：创建一个包含“书名”和“数量”两个字段的图书库存表

```
数据库连接.执行SQL("CREATE TABLE 图书库存表(书名 text(10),数量 int)")
```

### 6) “取得权限 ()” 命令

<整数型> 对象. 取得权限 ()

取得访问及相关权限，并返回一个整数代表用户连接当前的权限。该整数为以下常量值之一或之和：0（#未知权限）； 1（#只读权限）； 2（#只写权限）； 3（#读写权限）； 4（#独占读权限）； 8（#独占写权限）； 12（#独占权限）； 16（#共享权限）。





## 7) “设置权限 ()” 命令

<逻辑型> 对象. 设置权限 (整数型 要设置的模式)

设置访问及相关权限, 并返回一个逻辑型值, 如果设置成功, 即返回真, 如果设置不成功, 即返回假。

**注解:** 该命令只能在连接处于关闭状态的时候才可以使用。

参数<1>的名称为“要设置的模式”, 类型为“整数型 (int)”。要设置的模式为以下常量值之一或之和: 0 (#未知权限); 1 (#只读权限); 2 (#只写权限); 3 (#读写权限); 4 (#独占读权限); 8 (#独占写权限); 12 (#独占权限); 16 (#共享权限)。

## 8) “设超时时间 ()” 命令

<逻辑型> 对象. 设超时时间 (整数型 要设置的时间)

设置超时时间, 以毫秒为单位, 1000毫秒为1秒钟, 若时间超出, 会取消相关命令, 并弹出信息提示框。

参数<1>的名称为“要设置的时间”, 类型为“整数型 (int)”。本参数指定准备设置的时间 (毫秒)。

## 9) “取超时时间 ()” 命令

<整数型> 对象. 取超时时间 ()

取得连接超时时间, 以毫秒为单位, 1000毫秒为1秒钟。

## 10) “开始事务 ()” 命令

<整数型> 对象. 开始事务 ()

开始新事务, 返回一个整数型值, 该数值表示当前事务嵌套的层数, 譬如如果返回1, 则表明为顶层事务。

## 11) “保存事务 ()” 命令

<逻辑型> 对象. 保存事务 ()

保存任何更改并结束当前事务, 返回一个逻辑值, 如果保存事务成功返回真, 否则返回假。

## 12) “回滚事务 ()” 命令

<逻辑型> 对象. 回滚事务 ()

取消当前事务中所作的任何更改并结束事务, 返回一个逻辑值, 如果操作成功返回真, 否则返回假。

使用事务操作, 相当于对数据库操作过程的备份。“开始事务”后, 会记录下对数据库的一系列操作, 如果使用“回滚事务”, 那么在“开始事务”后对数据库的所有操作都会被取消。可以利用事务处理, 提高数据库操作的安全性。

如: 在对数据库进行一系列操作, 如果其中一个环节出现了错误, 就让数据库恢复到最初的样子。





在实际应用中，事务操作是非常重要的。例如银行系统的数据库操作需要非常高的安全性，从存储账户到支票账户的转账涉及多个数据库操作，只有这些数据库操作全部完成，才完成这次转账。在转账前新建一个事务；在转账的过程中，有任何一个环节出现错误，就回滚事务，取消前面对数据库的操作；只有所有操作都成功了，才“保存事务”。这样大大提高了对数据库操作的安全性。

数据库连接组件的事务操作命令与外部数据库组件的事务操作命令，虽然名称有所不同，但是操作流程是一样的。

### 6.4.3 “记录集”组件

#### 6.4.3.1 “记录集”组件的属性

##### 1) “是否已打开”属性

只读，数据类型：逻辑型，判断记录集是否打开。

##### 2) “操作状态”属性

只读，数据类型：整数型，取得当前操作状态。

##### 3) “记录数量”属性

只读，数据类型：整数型，取得记录的数量。如果为-1，表示ADO当前无法判断记录的数量，或者数据提供者不支持这个属性。

##### 4) “字段数量”属性

只读，数据类型：整数型，取得字段的数量。

##### 5) “首记录前”属性

只读，数据类型：逻辑型，判断当前记录指针是否已在首记录前。

##### 6) “尾记录后”属性

只读，数据类型：逻辑型，判断当前记录指针是否已在记录尾后。

##### 7) “当前位置”属性

只读，数据类型：整数型，当前记录在记录集中的位置，0为第一条记录。如果为-2表示位置未知（多数由于数据提供者（Provider）不支持IRowsetScroll接口），如为-3表示处于文件开头，如为-4表示文件结尾。

#### 6.4.3.2 “记录集”组件的命令

##### 1) “置连接（）”命令

<逻辑型> 对象. 置连接（数据库连接 连接）

设置数据库连接。

参数<1>的名称为“连接”，类型为“数据库连接”。本参数提供要设置的数据库连接对象。





例如：将记录集组件与数据库连接组件相关联。

记录集.置连接(数据库连接)

## 2) “取连接 ()” 命令

<数据库连接> 对象. 取连接 ()

取得数据库连接。

## 3) “打开 ()” 命令

<逻辑型> 对象. 打开 (文本型 命令文本, 整数型 命令类型)

打开数据库表，用于具体指定是操作哪一个表，返回一个逻辑值，如果打开表成功返回真，否则返回假。必须使用数据库连接组件先进行数据库连接，再用“置连接”命令连接到本对象，最后才能打开表。

参数<1>的名称为“命令文本”，类型为“文本型 (text)”。本参数可以提供数据库中的某一个数据表名或SQL语句。

参数<2>的名称为“命令类型”，类型为“整数型 (int)”。本参数提供指定命令文本的类型，可以为以下常量或数值之一：1 (#SQL语句)；2 (#数据表名)。

例如：打开图书库存表。两句代码的效果是一样的。

记录集.打开 (“图书库存表”, #数据表名)

记录集.打开 (“select \* from 图书库存表”, #SQL语句)

## 4) “打开并排序 ()” 命令

<逻辑型> 对象. 打开并排序 (文本型 表名, 文本型 排序条件)

打开数据表，并以指定条件排序。返回一个逻辑型值，如果排序成功返回真，否则返回假。

参数<1>的名称为“表名”，类型为“文本型 (text)”。本参数提供数据库表的名称。

参数<2>的名称为“排序条件”，类型为“文本型 (text)”。本参数指定排序条件，ASC为升序，DESC为降序。

例如：打开图书库存表，并按书号字段升序排列。

记录集.打开并排序 (“图书库存表”, “书号 ASC”)

## 5) “打开并过滤 ()” 命令

<逻辑型> 对象. 打开并过滤 (文本型 表名, 文本型 过滤条件)

用指定过滤条件打开一个表。

参数<1>的名称为“表名”，类型为“文本型 (text)”。本参数提供数据库表的名称。

参数<2>的名称为“过滤条件”，类型为“文本型 (text)”。过滤的条件。

例如：打开图书库存表，只包含表中价格大于60的记录。

记录集.打开并过滤 (“图书库存表”, “价格>60”)







## 6) “关闭 ()” 命令

<无返回值> 对象. 关闭 ()

关闭当前打开的数据库表，打开一个数据表，在使用后必须进行关闭操作。

例如：关闭打开的记录集。

记录集.关闭 ()

## 7) “添加 ()” 命令

<逻辑型> 对象. 添加 ()

添加一个新的空记录，返回一个逻辑值。如果添加成功返回真，否则返回假。

例如：在数据库中添加一条空记录。本命令在添加记录数据前使用。

记录集.添加 ()

## 8) “更新 ()” 命令

<逻辑型> 对象. 更新 ()

更新记录，刷新显示，返回一个逻辑值。如果更新成功返回真，否则返回假。

例如：添加记录数据后，要使用“更新”命令更新数据库内容。

记录集.更新 ()

## 9) “删除 ()” 命令

<逻辑型> 对象. 删除 (整数型 删除选项)

参数<1>的名称为“删除选项”，类型为“整数型 (int)”。本参数为以下常量或数值之一：1（删除当前记录）；3（删除全部记录）。

**注解：**删除全部记录仅对以“表名称”打开的记录集有效，对以“查询类SQL语句 (selete)”打开的记录集无效；可参考“打开”命令。

例如：删除当前记录。删除时要注意记录号是否准确。

记录集.删除 (1)

## 10 “写文本 ()” 命令

<逻辑型> 对象. 写文本 (通用型 序号或字段名, 文本型 文本)

通过序号或字段名写入文本。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型 (all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“文本”，类型为“文本型 (text)”。本参数提供所要写入的文本。

例如：添加记录时，加入书名字段的内容。

记录集.写文本 (“书名”，“唐诗三百首”)

## 11) “写单精度 ()” 命令

<逻辑型> 对象. 写单精度 (通用型 序号或字段名, 小数型 单精度)





通过序号或字段名写入单精度值。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“单精度”，类型为“小数型(float)”。本参数提供所要写入的小数。

例如：添加记录时，加入价格字段的内容。

```
记录集.写单精度("价格",81.12)
```

#### 12) “写整数 ()” 命令

<逻辑型> 对象. 写整数 (通用型 序号或字段名, 整数型 整型)

通过序号或字段名写入整数。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“整型”，类型为“整数型(int)”。本参数提供所要写入的整数。

例如：添加记录时，加入数量字段的内容。

```
记录集.写整数("数量",8)
```

#### 13) “写逻辑值 ()” 命令

<逻辑型> 对象. 写逻辑值 (通用型 序号或字段名, 逻辑型 逻辑)

通过序号或字段名写入逻辑值。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“逻辑”，类型为“逻辑型(bool)”。本参数提供所要写入的逻辑值。

例如：添加记录时，加入更换字段的内容。

```
记录集.写逻辑值("更换",真)
```

#### 14) “写日期时间 ()” 命令

<逻辑型> 对象. 写日期时间 (通用型 序号或字段名, 日期时间型 日期时间,  
[整数型 日期时间类型])

通过序号或字段名写入日期型数据。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。





参数<2>的名称为“日期时间”，类型为“日期时间型（date）”。本参数提供要写入的日期型数据。

参数<3>的名称为“日期时间类型”，类型为“整数型（int）”，可以被省略。本参数提供要写入的日期时间型数据的类型，为以下常量值之一：0（#日期时间型）；1（#日期型）；2（#时间型）。

**注解：**如果该值不为以上常量那么将使用#日期时间型。

例如：添加记录时，加入入库日期字段的内容。

记录集.写日期时间（“入库日期”，日期框.今天,1）

#### 15) “写双精度（）”命令

<逻辑型> 对象. 写双精度（通用型 序号或字段名，双精度小数型 双精度）

通过序号或字段名写入文本。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“双精度”，类型为“双精度小数型（double）”。本参数提供所要写入的双精度小数型。

例如：添加记录时，加入价格字段的内容。

记录集.写双精度（“价格”，423.56）

#### 16) “写字节集（）”命令

<逻辑型> 对象. 写字节集（通用型 序号或字段名，字节集型 数据）

通过序号或字段名写入数据。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“数据”，类型为“字节集型（bin）”。本参数提供所要写入的数据。

例如：添加记录时，加入数据字段的内容。

记录集.写字节集（“数据”，{1, 2, 3}）

#### 17) “读文本（）”命令

<逻辑型> 对象. 读文本（通用型 序号或字段名，文本型变量 文本变量）

通过序号或字段名读取文本数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“文本变量”，类型为“文本型（text）”，提供参数数据时只能提供变量。本参数提供要读取到的文本变量。





例如：读取记录时，将书名字段的内容放到文本变量中。

记录集.读文本(“书名”，文本变量)

#### 18) “读双精度 ()” 命令

<逻辑型> 对象. 读双精度 (通用型 序号或字段名, 双精度小数型变量 双精度变量)

通过序号或字段名读取双精度数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型 (all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“双精度变量”，类型为“双精度小数型 (double)”，提供参数数据时只能提供变量。本参数提供要读取到的双精度变量。

例如：读取记录时，将价格字段的内容放到双精度小数变量中。

记录集.读双精度(“价格”，双精度小数变量)

#### 19) “读单精度 ()” 命令

<逻辑型> 对象. 读文本 (通用型 序号或字段名, 小数型变量 单精度变量)

通过序号或字段名读取小数数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型 (all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“单精度变量”，类型为“小数型 (float)”，提供参数数据时只能提供变量。本参数提供要读取到的单精度变量。

例如：读取记录时，将价格字段的内容放到小数变量中。

记录集.读单精度(“价格”，小数变量)

#### 20) “读整数 ()” 命令

<逻辑型> 对象. 读整数 (通用型 序号或字段名, 整数型变量 整数变量)

通过序号或字段名读取整数数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型 (all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“整数变量”，类型为“整数型 (int)”，提供参数数据时只能提供变量。本参数提供要读取到的整数变量。

例如：读取记录时，将数量字段的内容放到整数变量中。

记录集.读整数(“数量”，整数变量)

#### 21) “读逻辑值 ()” 命令

<逻辑型> 对象. 读逻辑值 (通用型 序号或字段名, 逻辑型变量 逻辑变量)







通过序号或字段名读取逻辑数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“逻辑变量”，类型为“逻辑型(bool)”，提供参数数据时只能提供变量。本参数提供要读取到的逻辑变量。

例如：读取记录时，将更换字段的内容放到逻辑变量中。

记录集.读逻辑值(“更换”，逻辑变量)

## 22) “读日期时间 ()” 命令

<逻辑型> 对象. 读日期时间 (通用型 序号或字段名, 日期时间型变量 日期变量)

通过序号或字段名读取日期时间数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“日期变量”，类型为“日期时间型(date)”，提供参数数据时只能提供变量。本参数提供要读取到的日期变量。

例如：读取记录时，将入库日期字段的内容放到日期变量中。

记录集.读日期时间(“入库日期”，日期变量)

## 23) “读字节集 ()” 命令

<逻辑型> 对象. 读字节集 (通用型 序号或字段名, 字节集型变量 字节集变量)

通过序号或字段名读取字节集数据到变量中。返回一个逻辑型值，如果写成功返回真，否则返回假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

参数<2>的名称为“字节集变量”，类型为“字节集型(bin)”，提供参数数据时只能提供变量。本参数提供要读取到的字节集变量。

例如：读取记录时，将数据字段的内容放到字节集变量中。

记录集.读字节集(“数据”，字节集变量)

## 24) “保存到XML ()” 命令

<逻辑型> 对象. 保存到XML (文本型 XML文件名)

将数据库表保存到XML文本文件，以作为交换通用数据表，返回一个逻辑值，如果保存XML文件成功返回真，否则返回假。

参数<1>的名称为“XML文件名”，类型为“文本型(text)”。本参数提供要写出到的XML文件名。





## 25) “保存到ADTG ()” 命令

<逻辑型> 对象. 保存到ADTG (文本型 ADTG文件名)

将数据库表用专用的“Advanced Data Tablegram”格式保存, 返回一个逻辑值, 如果保存成功返回真, 否则返回假。

参数<1>的名称为“ADTG文件名”, 类型为“文本型(text)”。本参数提供要写出到的ADTG文件名。

## 26) “到首记录 ()” 命令

<逻辑型> 对象. 到首记录 ()

移动到表的第一条记录, 返回一个逻辑值, 如果移动成功返回真, 否则返回假。操作本命令之前要确保数据库及表打开。

## 27) “到尾记录 ()” 命令

<逻辑型> 对象. 到尾记录 ()

移动到表的最后一条记录, 返回一个逻辑值, 如果移动成功返回真, 否则返回假。操作本命令之前要确保数据库及表打开。

## 28) “到下一条 ()” 命令

<逻辑型> 对象. 到下一条 ()

移动到下一条记录, 返回一个逻辑值, 如果移动成功返回真, 否则返回假。操作本命令之前要确保数据库及表打开。

## 29) “到前一条 ()” 命令

<逻辑型> 对象. 到前一条 ()

移动到前一条记录, 返回一个逻辑值, 如果移动成功返回真, 否则返回假。操作本命令之前要确保数据库及表打开。

“到首记录 ()”、“到尾记录 ()”、“到下一条 ()”、“到前一条 ()”命令都是用来控制当前记录指针的。

## 30) “移到 ()” 命令

<逻辑型> 对象. 移到 (整数型 目标记录号)

移到指定记录。

参数<1>的名称为“目标记录号”, 类型为“整数型(int)”。本参数提供目标记录号值, 该值从0开始。

例如: 将记录指针移到第5条记录。

记录集.移到(4)

## 31) “取字段名 ()” 命令

<文本型> 对象. 取字段名 (整数型 字段序号)

取得指定序号字段的名称, 返回一个文本型值, 为字段名称文本。







参数<1>的名称为“字段序号”，类型为“整数型（int）”。本参数提供当前表中指定字段的序号值，从0开始。

例如：在输出面板中输出第一个字段的名称。

输出调试文本(记录集.取字段名(0))

### 32) “取字段属性 ()” 命令

<整数型> 对象. 取字段属性 (通用型 序号或字段名)

通过序号或字段名取得字段属性，返回一个整数值。为以下常量值之一或之和：  
2（#字段被延迟）； 4（#字段可写入）； 8（#写入无法确定）； 16（#定长数据）；  
32（#允许空值）； 64（#可以读空值）； 128（#二进制类型）； 256（#持久标识符）；  
512（#包含标记）； 4096（#字段被缓存）。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

例如：在输出面板中输出第一个字段的属性。

输出调试文本(记录集.取字段属性(0))

### 33) “取字段定义长度 ()” 命令

<整数型> 对象. 取字段定义长度 (通用型 序号或字段名)

通过序号或字段名取得字段定义长度，本命令返回一个整数值。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

例如：在输出面板中输出第一个字段的定义长度。

输出调试文本(记录集.取字段定义长度(0))

### 34) “取字段实际长度 ()” 命令

<整数型> 对象. 取字段实际长度 (通用型 序号或字段名)

通过序号或字段名取得字段实际长度，返回一个整数型值。

参数<1>的名称为“序号或字段名”，类型为“通用型（all）”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

例如：在输出面板中输出第一个字段的实际长度。

输出调试文本(记录集.取字段实际长度(0))

### 35) “取字段类型 ()” 命令

<整数型> 对象. 取字段类型 (通用型 序号或字段名)

通过序号或字段名取得字段类型，返回以下常量值之一： 20（#超长整数字段）；  
128（#二进制字段）； 11（#逻辑型字段）； 8（#宽文本字段1）； 129（#字符型字段）；  
6（#货币型字段）； 7（#日期型字段）； 133（#数据库日期型字段）； 134（#时间型  
字段）； 135（#时间戳字段）； 14（#数值型字段）； 5（#双精度字段）； 0（#空白型





字段)； 10 (#错误码字段)； 72 (#标识型字段)； 9 (#IDispatch字段)； 3 (#整数型字段)； 13 (#IUnknown字段)； 205 (#超变长二进制字段)； 201 (#超变长文本字段)； 203 (#超变长宽文本字段)； 131 (#数字型字段)； 4 (#单精度字段)； 2 (#短整数字段)； 16 (#有符号字节字段)； 21 (#无符号超长整数字段)； 19 (#无符号整数字段)； 18 (#无符号短整数字段)； 17 (#字节型字段)； 132 (#自定义型字段)； 204 (#变长二进制字段)； 200 (#变长文本字段)； 12 (#变体型字段)； 202 (#变长宽文本字段)； 130 (#宽文本字段2)。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

例如：在输出面板中输出第一个字段的数据类型。

输出调试文本(记录集.取字段类型(0))

### 36) “字段是否为空 ()” 命令

<逻辑型> 对象. 字段是否为空 (通用型 序号或字段名)

判断指定字段是否为空。返回一个逻辑型值，如果为空即为真，否则为假。

参数<1>的名称为“序号或字段名”，类型为“通用型(all)”。本参数提供字段的序号或者字段名，只能为数值或文本型，序号值从0开始。

### 37) “查找 ()” 命令

<逻辑型> 对象. 查找 (文本型 查找条件, 整数型 查找方向)

查找满足条件的记录，返回一个逻辑型值，如果找到记录即返回真，并停留在满足条件的记录上，否则返回假。找到第一个满足条件的记录后，以后可以用”查找下一个”命令继续进行查找。

参数<1>的名称为“查找条件”，类型为“文本型(text)”。本参数提供在数据表中查找的条件。

参数<2>的名称为“查找方向”，类型为“整数型(int)”。本参数指定在数据表中查找的方向。可以为以下数值或常量之一：1（正向搜索）；-1（反向搜索）。

例如：正向查找满足条件的记录。

记录集.查找(“出版社=’易语言出版社’”,1)

### 38) “查找下一个 ()” 命令

<逻辑型> 对象. 查找下一个 ()

查找下一个满足条件的记录。此命令用于”查找”命令之后使用，可继续按查找命令找到下一个满足条件的记录。返回一个逻辑型值，如果找到返回真，否则返回假。

### 39) “过滤记录 ()” 命令

<逻辑型> 对象. 过滤记录 (文本型 过滤条件)

过滤出满足条件的记录，返回一个逻辑型值，如果过滤成功返回真，否则返回假。





**注解：**在不使用过滤的时候，一定要再次调用本函数并将过滤条件指定为空文本。

参数<1>的名称为“过滤条件”，类型为“文本型(text)”。本参数提供过滤的条件。

例如：过滤满足条件的记录。

```
记录集.过滤记录("入库日期='2009年1月19日'")
```

'在此可以显示过滤后的记录

```
记录集.过滤记录("")
```

至此，易语言对外部数据库的两种访问方式基本介绍完毕。以后，在用到外部数据库时，要根据数据库的类型和使用数据库的环境，确定具体使用哪一种访问方式操作数据库。

## 6.5 小结

易语言支持国际通用的ODBC和ADO数据库访问协议。基本上所有的外部大型数据库，如MYSQL、SQL Server、Oracle、Sybase等数据库都可以被易语言程序所访问。SQL语句是关系型数据库的标准化规范语言，易语言支持使用SQL语句操作外部数据库。

## 6.6 习题

- 6-1 易语言访问外部数据库的两种方式：\_\_\_\_\_和\_\_\_\_\_。
- 6-2 SQL语言是\_\_\_\_\_。
- 6-3 SQL查询语句的关键字是：\_\_\_\_\_。
- 6-4 记录集中，记录号从\_\_\_\_\_开始。
- 6-5 易语言核心支持库中的数据库组件有\_\_\_\_\_、\_\_\_\_\_。





## 第四部分 易语言高级应用

前几章学习了易语言编程的基础知识，还详细介绍了易语言命令和组件的使用，以及易语言对数据库的操作。那么易语言还有哪些功能可以在编程中带来方便呢？

在本部分中，将学习、了解易语言中模块、向导、API、DLL、COM对象、OCX组件、类型库等编程资源的应用；还有易语言的面向对象编程及易语言对程序的调试和编译的过程。





## 第七章 DLL的应用

### 本章目标

在本章结束时，我们能够：

- 了解什么是DLL
- 了解编写一个DLL的步骤
- 学会编译发布完成的DLL
- 学会用易语言调用DLL
- 学会用其他语言调用易语言编写的DLL



## 7.1 了解DLL

DLL即动态链接库 (Dynamic Link Library)，它允许程序共享执行特殊任务所必需的代码和资源。Windows提供的DLL文件中包含了允许基于Windows的程序在Windows环境下操作的许多命令和资源。

在Windows操作系统中，DLL对于程序执行是非常重要的，因为程序在执行的时候，必须链接到DLL文件才能够正确地运行。而有些DLL文件可以被许多程序共用。

使用DLL动态链接库的一些好处：

(1) 多个应用程序共享代码和数据：比如Office软件的各个组成部分有相似的外观和功能，这就是通过共享DLL动态链接库实现的。

(2) 某些程序过滤系统消息时必须使用DLL动态链接库。

(3) DLL动态链接库以一种自然的方式将一个大的应用程序划分为几个小的模块，有利于小组内成员的分工与合作。而且，各个模块可以独立升级。如果小组中的一个成员开发了一组实用例程，他就可以把这些例程放在一个动态链接库中，让小组的其他成员使用。

(4) 为了实现应用程序的国际化，往往需要使用动态链接库。使用DLL动态链接库可以将针对某一国家、语言的信息存放在其中。对不同的版本，使用不同的DLL动态链接库。在使用应用程序向导生成应用程序时，可以指定资源文件使用的语言，这就是通过提供不同的DLL动态链接库实现的。

DLL不是独立运行的程序，它是某个程序的一个部分，只能由所属的程序调用。

VC++、C++ Builder、Delphi都可以编写DLL文件。Visual Basic 5.0以上版本也可以编写一种特殊的DLL，即ActiveX DLL。同样，易语言也能编写DLL文件并封装自己编制的命令，供其他编程工具调用，由此实现与其他编程工具的互通交流。由于国外的编程工具对中文的处理实现相对繁琐，但通过使用易语言封装一些中文处理命令，将极大地方便编程人员的工作。

## 7.2 编写DLL

易语言从3.6版开始支持对DLL动态链接库的开发，编译出的DLL是标准的WIN32 DLL文件，可供其他编程语言工具调用，如VB、VC、Delphi等。

通过一个简单的例程向大家介绍DLL的编写方法。







目标：通过调用自定义DLL文件中的命令来计算任意三角形的面积。  
第一步：启动易语言，新建一个“Windows动态链接库”程序，如图7-1所示。

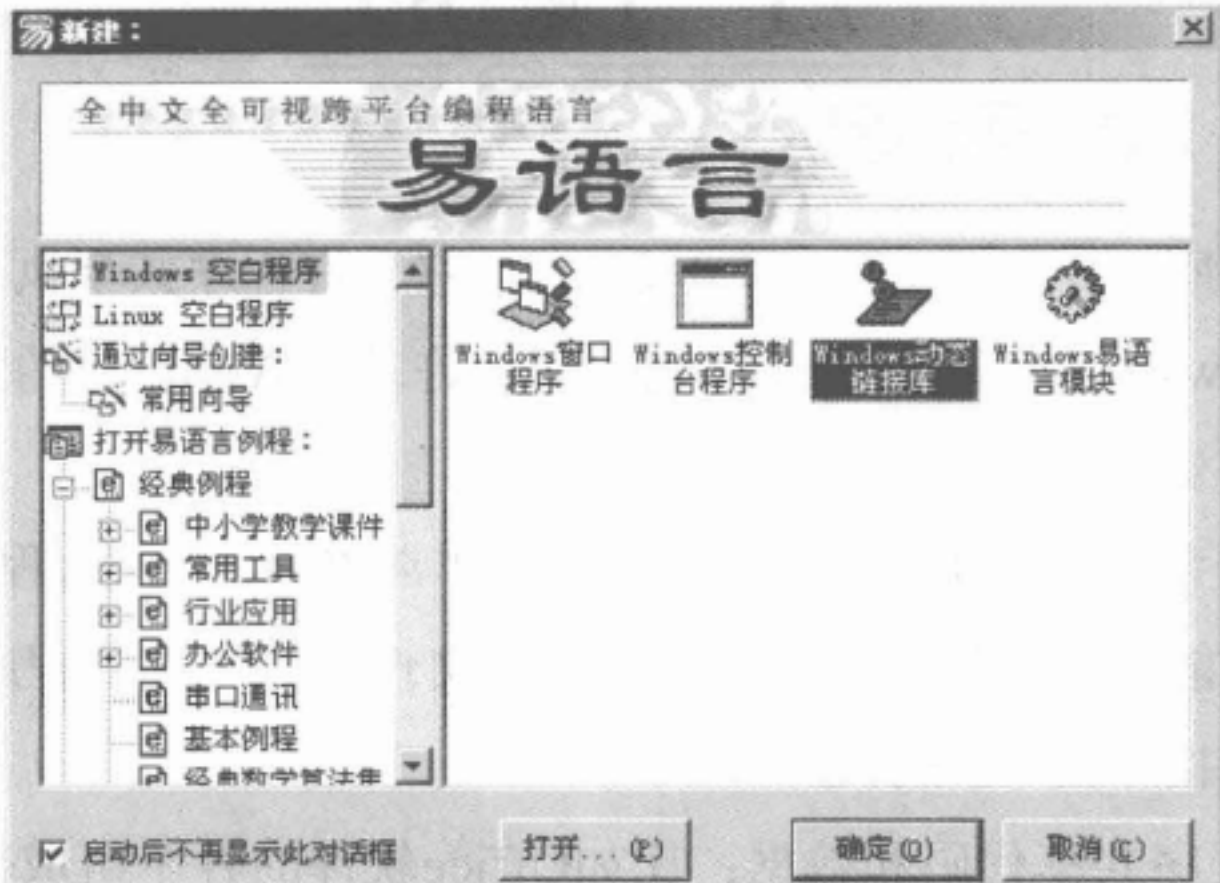


图7-1 新建DLL文件

在易语言代码编辑区将自动生成如下代码：

程序集名	保留	保留	备注
程序集1			

子程序名	返回值类型	公开	备注
_启动子程序	整数型		请在本子程序中放置动态链接库初始化代码

\_临时子程序 0 ' 在初始化代码执行完毕后调用测试代码  
返回 (0) ' 返回值被忽略。

子程序名	返回值类型	公开	备注
_临时子程序			

' 本名称子程序用作测试程序用，仅在开发及调试环境中有效，编译发布程序前将被系统自动清空，请将所有用作测试的临时代码放在本子程序中。 \*\*\*注意不要修改本子程序的名称、参数及返回值类型。

第二步：新建一个子程序，然后把“公开”选中。

子程序名	返回值类型	公开	备注
子程序1		✓	

在DLL文件中，任何程序集中选中“公开”的子程序都可作为对外接口使用。此子程序的名称就是需要调用的易语言DLL文件中的命令的名称。

第三步：更改子程序名称为“求任意三角形面积”，然后创建三个参数。代码如下：

子程序名	返回值类型	公开	备 注		
求任意三角形面积	小数型	✓			
参数名	类 型	参考	可空	数组	备 注
边长一	小数型				
边长二	小数型				
边长三	小数型				



第四步：编写如下代码：

子程序名	返回值类型	公开	备注		
求任意三角形面积	小数型	✓			
参数名	类型	参考	可空	数组	备注
边长一	小数型				
边长二	小数型				
边长三	小数型				

变量名	类型	静态	数组	备注
半周长	小数型			
乘积	小数型			
面积	小数型			

半周长 = (边长一 + 边长二 + 边长三) ÷ 2  
乘积 = 半周长 × (半周长 - 边长一) × (半周长 - 边长二) × (半周长 - 边长三)  
面积 = 求平方根 (乘积)  
返回 (四舍五入 (面积, 2)) ' 保留两位小数

这样求任意三角形面积.DLL代码的编写就完成。

### 7.3 编译DLL

本节将通过上一节完成的DLL代码来讲解DLL的编译。

选择菜单“编译”→“编译”或按F7键将其编译为DLL，修改文件名为“求任意三角形面积”，编译后的DLL文件后缀名是“.dll”，如图7-2所示。

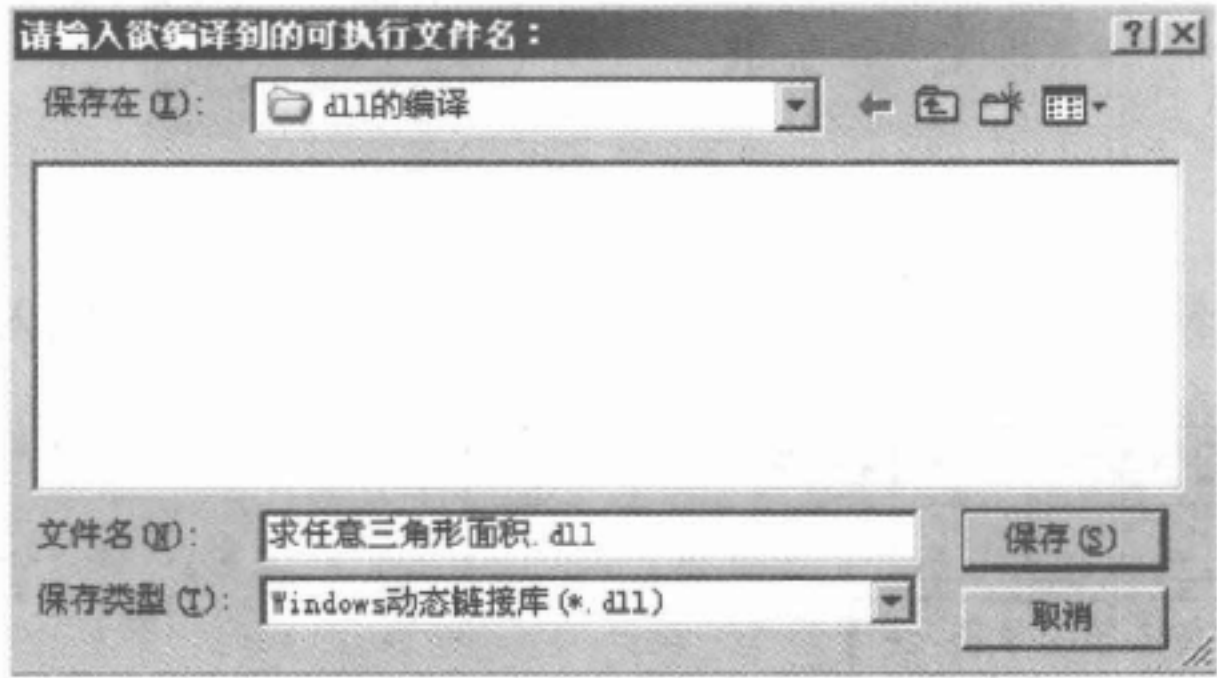


图7-2 保存编译后的DLL

保存后，易语言系统会提示：是否将涉及到的相关支持库文件写到DLL文件所在的目录，并且会在易语言的“输出”面板中提示DLL文件运行时所依赖的支持库文件列表。

易语言5.X版本增加了对DLL文件的静态编译功能，而静态编译的DLL文件，其他编程语言在调用时是不需要依赖易语言支持库的。





## 7.4 调用DLL

用易语言编写的DLL不仅可以被易语言调用，也可以被其他编程语言（如VB、VC、Delphi等）调用。

下面来看一下在易语言中的调用方法：

第一步：新建一个易程序，再新建一个DLL命令，定义DLL命令。

Dll命令名	返回值类型	公开	备 注	
求任意三角形面积	小数型			
Dll库文件名：				
求任意三角形面积.dll				
在Dll库中对应命令名：				
求任意三角形面积				
参数名	类 型	传址	数组	备 注
边长一	小数型			
边长二	小数型			
边长三	小数型			

第二步：在窗体中添加按钮组件、标签组件、编辑框组件，设计程序界面如图7-3所示。

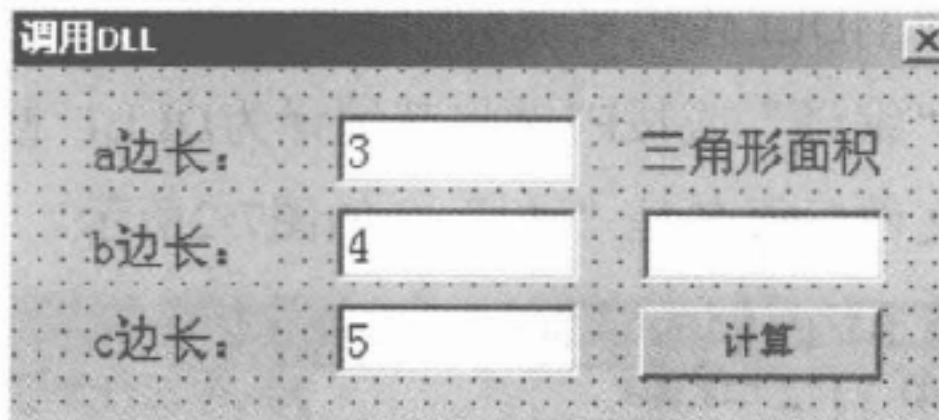


图7-3 调用DLL命令的窗口程序

第三步：双击按钮组件，在“\_按钮1\_被单击”事件子程序中输入以下代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
a边长	小数型			
b边长	小数型			
c边长	小数型			

a边长 = 到小数 (编辑框1.内容)

b边长 = 到小数 (编辑框2.内容)

c边长 = 到小数 (编辑框3.内容)

面积编辑框.内容 = 到文本 (求任意三角形面积 (a边长, b边长, c边长))





把此程序保存在与所用到的“求任意三角形面积.dll”文件相同的目录下。

第四步：按F5键运行程序。单击按钮，通过调用DLL中的接口命令计算任意三角形的面积，如图7-4所示。

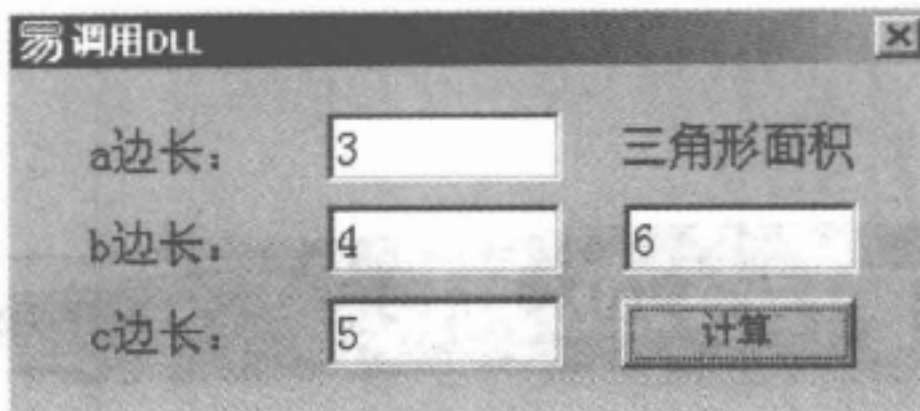


图7-4 运行后的结果

**注解：**在其他编程语言中应用此DLL文件时，一定要将核心支持库文件“krnlfnr”随程序一同发布，如果使用易语言5.X版本静态编译DLL，则不用将易语言核心支持库随程序一同发布。

## 7.5 小结

DLL动态链接库文件是一个拥有多个命令集合的文件，就是将一些能够公用的命令集成在一起的一个Windows标准接口的命令的集合。DLL接口命令的类型是文本型或字节集型时，要使用指针来传递数据。定义DLL命令时，“DLL库文件名”中所指定的DLL文件路径要正确。DLL文件可以简化代码量，使编程更加方便快捷。在易语言中，可以创建标准的动态链接库文件（DLL）供其他编程语言调用。同时，易语言也可以调用由其他编程语言编写的动态链接库，从而达到了编程功能互通、互补的目的。

## 7.6 习题

- 7-1 DLL文件即\_\_\_\_\_文件。
- 7-2 发布易语言编写的DLL文件必须\_\_\_\_\_。
- 7-3 DLL命令中返回值类型是文本型或字节集型时，调用返回值必须使用\_\_\_\_\_和\_\_\_\_\_命令加以转换。
- 7-4 定义DLL命令时，“DLL库文件名”中所指定的DLL文件必须位于\_\_\_\_\_。
- 7-5 尝试用“取汉字拼音（）”命令编写一个DLL。
- 7-6 编写一个含有窗口的DLL。





## 第八章 API的应用

### 本章目标

在本章结束时，我们能够：

- 了解什么是API
- 了解在易语言中如何定义API
- 学会看懂API手册中的说明
- 学会正确调用API



## 8.1 了解API

API (Application Programming Interface, 应用程序编程接口) 是软件厂商为二次开发提供的一组命令。使用API可以方便地调用其他程序所提供的功能。

WIN32 API是Microsoft Windows 32位平台的应用程序编程接口。对这个定义的理解,需要追溯到操作系统的发展历史上,当Windows操作系统开始占据主导地位的时候,开发Windows平台下的应用程序成为人们的需要。而在Windows程序设计领域处于发展的初期,Windows程序员所能使用的编程工具唯有API命令,这些命令是Windows提供给应用程序与操作系统间的接口,他们犹如“积木块”一样,可以搭建出各种界面丰富,功能灵活的应用程序。所以可以认为WIN32 API命令是构筑整个Windows框架的基石,在它的下面是Windows的操作系统核心,而它的上面则是所有的华丽的Windows应用程序。但是,那时的Windows程序开发还是比较复杂的工作,程序员必须熟记一大堆常用的WIN32 API命令,而且还得对Windows操作系统有深入的了解。

然而随着软件技术的不断发展,在Windows平台上出现了很多优秀的可视化编程环境,程序员可以采用“所见即所得”的编程方式来开发具有精美用户界面和功能强大的应用程序。这些优秀可视化编程环境操作简单、界面友好(诸如VB、易语言、Delphi等),在这些工具中提供了大量的类库和各种控件,它们替代了API的神秘功能,事实上这些类库和控件都是构架在WIN32 API命令基础之上的,是封装了的API命令的集合。它们把常用的API命令组合在一起成为一个控件或类库,并赋予其方便的使用命令,所以极大地加速了Windows应用程序开发的过程。有了这些控件和类库,程序员便可以把主要精力放在程序整体功能的设计上,而不必过于关注技术细节。

实际上如果要开发更灵活、实用、更具效率的应用程序,必然要涉及到直接使用API命令。虽然使用类库和控件开发应用程序简单得多,但它们只提供Windows的一般功能,对于比较复杂和特殊的功能来说,使用类库和控件是非常难以实现的,这时就需要采用API命令来实现。这也是API命令使用的场合,所以对待API命令不必刻意来研究每一个命令的用法,那也是不现实的(能用得到的API命令有几千个)。只是在需要的时候去查API帮助就足够了。

作为一个编程初学者来说,API命令也许是一个时常耳闻却感觉有些神秘的东西。单看它的复杂语法,就足以令人望而生畏。但是任何事物在深入了解它之前,总是会有这种感觉的。在本章将教大家怎么样在易语言中使用API,使它成为编程的好助手。







## 8.2 定义API

首先，一同来了解一下，易语言中是如何定义API命令的。

在Windows中，API命令一般位于DLL（动态链接库）文件中。在易语言里为了便于编程人员的理解，把调用API命令称之为“DLL命令”。

在“程序”面板中双击“DLL命令”选项，然后在代码编辑区中单击鼠标右键，在菜单中选择“新DLL命令”菜单项，如图8-1所示。

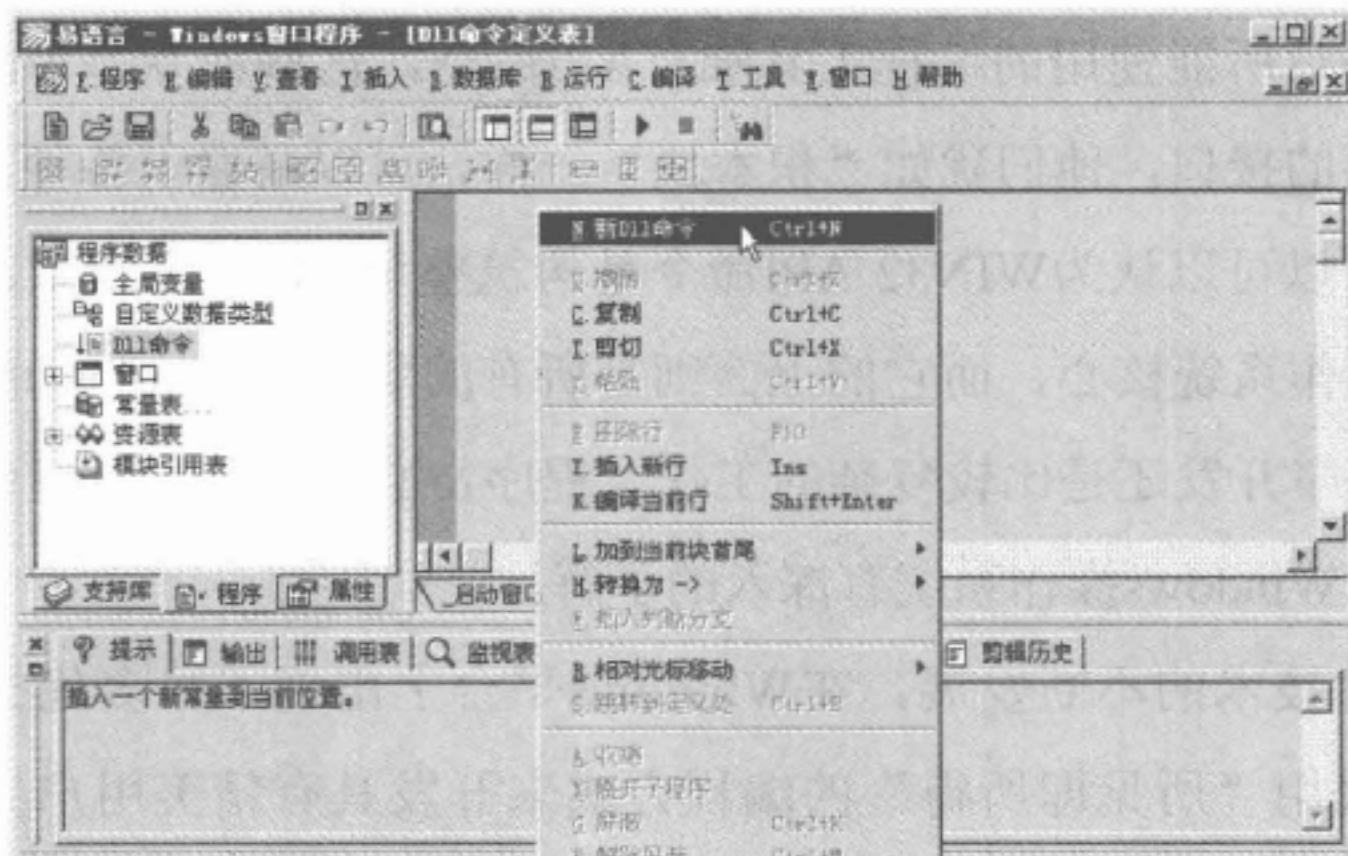


图8-1 新建DLL命令

或者在双击“DLL命令”选项之后，选择菜单“插入”→“现行单元”或按下Ctrl+N快捷键，同样可以新建一个DLL命令。

新建后，会在代码编辑区中出现一个DLL命令的表格式定义框架，如图8-2所示。

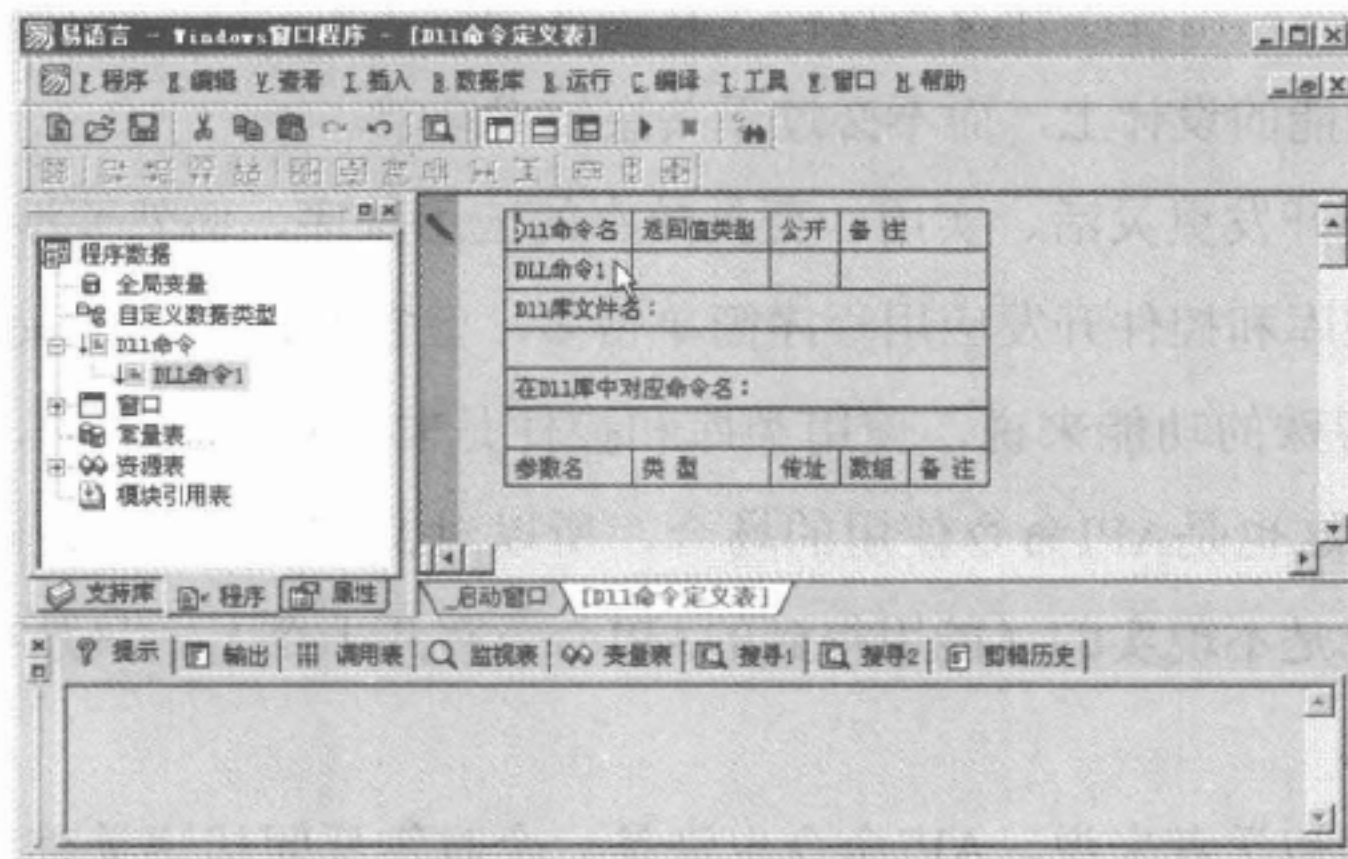


图8-2 DLL命令定义框架

只要根据需要，填写表格内容，即可完成DLL命令的定义。





DLL命令的定义说明:

- DLL命令名: 输入或修改当前 DLL命令在易程序中的使用名称。
- 返回值类型: 定义DLL命令返回值的数据类型, 不能是非系统基本数据类型或字节集型。
- 公开: 决定当前DLL命令在编译后是否对外公开以便外部可以直接调用 (一般应用于易模块)。

**备注:** 输入或修改与当前DLL命令相关的备注信息。

- DLL库文件名: 定义当前DLL命令所在动态链接库的文件名。如果不指定库文件名, 系统将默认在Kernel32.dll、Gdi32.dll、User32.dll、Mpr.dll、Advapi32.dll、Shell32.dll等Windows系统的基本WIN32 API库中搜寻指定命令。

**注解:** 这里最好不要包含DLL库文件名的路径, 程序运行后会自动定位该DLL库 (查找顺序为: 执行文件所在目录, 当前目录, 系统目录, 系统安装目录……)。如果使用的不是Windows系统自带的DLL文件, 且该DLL文件所在的目录又不是易语言要查找的目录, 那么DLL库文件名就要包含路径 (绝对路径: 如“D:\文件.dll”; 相对路径: 如“目录名\文件.dll”或“文件.dll”)。一般情况下, 最好将DLL文件与执行程序放在同一目录下, 使用相对路径。

- 在DLL库中对应命令名: 定义当前DLL命令在其所在动态链接库中的名称, 如果不指定, 系统将默认等同于“DLL命令名”。
- 参数名: 输入或修改当前DLL命令参数的名称, 这里可自行命名。
- 类型: 所定义的数据类型应该与实际的DLL库命令一致, 但不能为窗口、窗口组件、菜单。
- 传址: 某些实际 DLL库命令的参数需要接收数据地址, 设置本属性为真可以确保将参数数据的地址传递过去。如果参数数据的类型为数组、文本、字节集、用户自定义数据类型、库定义数据类型, 则无论此属性是否为真, 都将传递数据地址。如果本属性为真且调用 DLL命令时所传递过来数据的类型与相应位置处所定义参数的数据类型不一致, 但可以相互转换, 则系统先分配一段临时内存空间, 然后进行转换并将转换后的数据存放到该空间, 最后将此内存空间的地址传递给实际的 DLL库命令。
- 数组: 设置当前DLL命令参数是否为接收数组数据。

下面, 来看一个具体定义API的例子。

Windows系统中的Win32 API命令有很多, 网络上也有很多Win32 API命令的中文帮助文档, 里面列举了常用的Win32 API命令的相关资料。从Win32 API的帮助文档中任意选取一个Win32 API命令进行定义。此Win32 API命令的帮助说明如图8-3所示。

图8-3的VB声明 `Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long`中:







GetWindowText	
VB声明	
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long	
说明	
取得一个窗体的标题(caption)文字, 或者一个控件的内容(在vb里使用: 使用vb窗体或控件的caption或text属性)	
返回值	
Long, 复制到lpString的字符串长度; 不包括空中止字符。会设置GetLastError	
参数表	
参数	类型及说明
hwnd	Long, 欲获取文字的那个窗口的句柄
lpString	String, 预定义的一个缓冲区, 至少有cch+1个字符大小; 随同窗口文字载入
cch	Long, lpString缓冲区的长度
注解	
不能用它从另一个应用程序的编辑控件中获取文字	

图8-3 API的帮助说明

- “GetWindowText”：API命令的名称。
- “user32”：API命令所在的DLL文件名称，完整名称是“user32.dll”，因为是系统的DLL文件，所以不用写完整的路径。
- “GetWindowTextA”：在DLL库中对应的API命令名。

括号里的内容是API命令的参数解释：

- “ByVal”：表示以“传值”（区别于“Byref”“传址”）方式传递参数；
- “hwnd”：表示参数名称；
- “As Long”：表示参数的数据类型。

括号外面最后的“As Long”是API命令返回值的数据类型。

在定义API时，一般都根据API帮助文档中的VB声明来填写定义的内容。

API在易语言中的DLL命令的定义如下所示：

Dll命令名	返回值类型	公开	备 注	
取窗口标题_	整数型		取得一个窗体的标题	
Dll库文件名：				
user32				
在Dll库中对应命令名：				
GetWindowTextA				
参数名	类 型	传址	数组	备 注
窗口句柄	整数型			欲获取文字的那个窗口的句柄
缓冲区	文本型	✓		预定义的一个缓冲区
缓冲尺寸	整数型			缓冲区的长度：

API命令的定义过程中，会发现即使在API中文帮助文档中也会存在很多的英文解释。特别是API的参数类型，很少有中文说明。下面就列举一下VB中常用的API参数的数据类型与易语言数据类型的定义对照，见表8-1。





表8-1 VB版API参数的基本类型

参数类型	易语言中的定义
Byte	字节型，Byte翻译过来为字节型
String	文本型，一般使用时需要传址
Long	整数型，也有可能是其他的数值类型
Any	任意型，根据使用情况或其他参数的值决定；一般使用时需要传址
Rect	矩形，一种结构型数据类型，其他如：坐标结构POINTAPI，时间结构SYSTEMTIME等都属于结构型数据类型。在易语言的“自定义数据类型”里定义

以上为一般情况下API参数类型在易语言中的定义，其他特殊情况如：参数类型是Long，定义时要定义为子程序指针；参数类型是String，定义时要定义为整数型等，复杂的类型定义需要对API命令加深学习和应用，才能了解和掌握。

前面定义的API命令是根据VB中的API命令声明来定义的，而VB中的API命令声明是以MSDN（微软的技术资源库）中的说明为基准的。那么，易语言定义API命令为什么不直接根据MSDN中的说明来定义呢？这主要因为MSDN里全是英文资料，就算是一个英文非常好的人也不一定完全看懂MSDN里的内容；而VB对MSDN的内容做了简化，并按照一定的格式做了相关解释，这样能直观地看到API命令的结构和使用方法。因此在易语言中采用了VB声明API命令的方式加以对照定义。

例如：GetWindowText命令在MSDN中的定义：

```
int GetWindowText(  
    HWND hWnd,  
    LPTSTR lpString,  
    int nMaxCount  
);
```

粗体部分的“HWND”、“LPTSTR”、“int”是参数的数据类型，斜体部分是参数名称。命令最前面的“int”是返回值类型。

如果感兴趣也可通过网络在MSDN（<http://www.msdn.net/>）中搜索到GetWindowText命令，查看其全部说明。

MSDN中的某些常用的有符号和无符号的数据类型在易语言中的定义是一样的。见表8-2。

表8-2 MSDN版API参数的基本类型

MSDN参数类型	易语言中的定义	注意事项
Char 可以引申出lpstr（文本）类型	文本型，一般使用时需要传址	本身是字节型，使用字节型数组的方式保存文本指针
Long、int	整数型	注意类型长度：4位字节用整数型，8位字节用长整数型。
bool	逻辑型	
Float、double	小数型、双精度小数型	





## 8.3 调用API

易语言即可以调用Windows系统自带的API命令，也可以调用Windows系统外部扩展API命令。在了解了怎样定义API命令后，再来学习怎样调用它们。

### 8.3.1 调用系统API

**【例】** 一个延时关闭窗口的例程。

本例程中，通过调用API实现程序延迟运行的效果，与易语言中“延时”的命令很相似。

第一步：新建一个易程序，在“\_启动窗口”中添加一个按钮组件和一个标签组件，修改对应的名称及内容，如图8-4所示。

第二步：新建DLL命令，添写相应的API命令信息。本例程中将用到一个API命令：Sleep。

命令的说明：Sleep：延迟指定的时间，参数为时间数值，单位为毫秒。

API定义如下：

Dll命令名	返回值类型	公开	备 注	
延迟时间	整数型		Sleep	
Dll库文件名：				
kernel32.dll				
在Dll库中对应命令名：				
Sleep				
参数名	类 型	传址	数组	备 注
延迟毫秒数	整数型			dwMilliseconds

双击“延时关闭窗口”按钮，进入代码编辑区。输入如下代码：

子程序名	返回值类型	公开	备注
_延时按钮_被单击			
延时按钮.标题 = “还有 5 秒窗口关闭”			
延迟时间 (1000) / 延时第1秒			
延时按钮.标题 = “还有 4 秒窗口关闭”			
延迟时间 (1000) / 延时第2秒			
延时按钮.标题 = “还有 3 秒窗口关闭”			
延迟时间 (1000) / 延时第3秒			
延时按钮.标题 = “还有 2 秒窗口关闭”			
延迟时间 (1000) / 延时第4秒			
延时按钮.标题 = “还有 1 秒窗口关闭”			
延迟时间 (1000) / 延时第5秒			
_启动窗口.销毁 0			

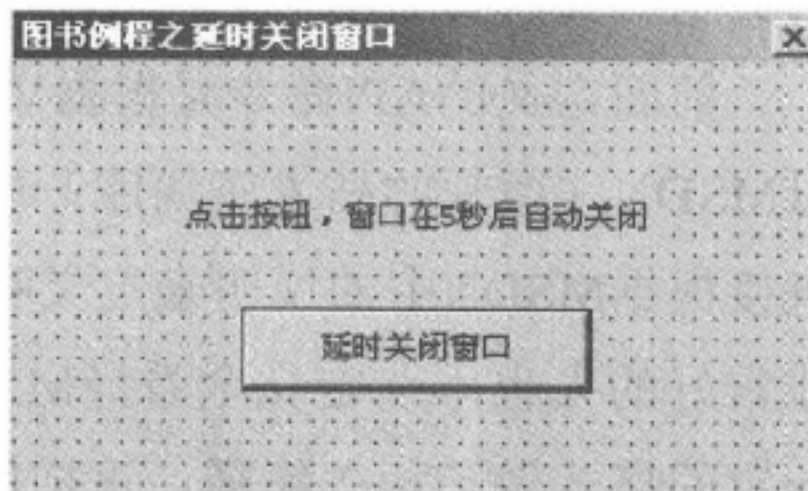


图8-4 “延时关闭窗口”程序界面





为了使程序的演示更清晰简洁，代码中分步骤显示延迟时间，每次延迟1秒。  
按F5运行程序。单击按钮后发现窗口并没有马上关闭，而是延迟了5秒后才关闭。

**【例】** 通过调用API命令，实现取鼠标所在窗口标题。

调用API命令，可以实现很多特殊的功能，这里演示一下通过使用两个API命令实现取鼠标所在窗口的标题。

第一步：新建一个易程序，在“\_启动窗口”中添加一个标签组件和一个时钟组件，如图8-5所示。

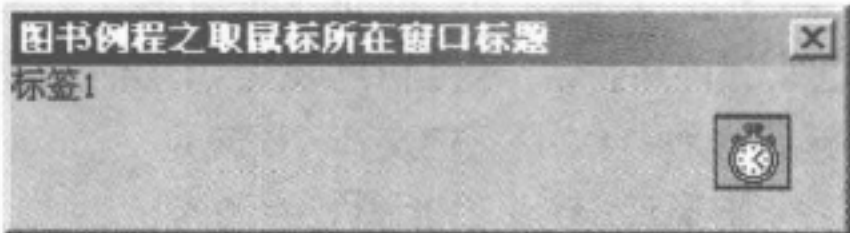


图8-5 “取鼠标所在窗口标题”程序界面

设置“时钟”的“时钟周期”属性为10，即每10毫秒运行一次取窗口标题的代码；“\_启动窗口”的“总在最前”属性为真；“标签1”的宽度和高度等于窗口客户区的宽度和高度。

第二步：新建DLL命令，添加API命令相关申明。

WindowFromPoint：取指定点所在窗口的句柄。

GetWindowTextA：取窗口的标题文字。

添加的两个API命令如下：

Dll命令名	返回值类型	公开	备 注	
取鼠标所在窗口句柄	整数型			
Dll库文件名：				
user32.dll				
在Dll库中对应命令名：				
WindowFromPoint				
参数名	类 型	传址	数组	备 注
横坐标	整数型			
纵坐标	整数型			

Dll命令名	返回值类型	公开	备 注	
取鼠标所在窗口标题	整数型			
Dll库文件名：				
user32. dll				
在Dll库中对应命令名：				
GetWindowTextA				
参数名	类 型	传址	数组	备 注
窗口句柄	整数型			
内存空间	文本型	✓		
内存空间大小	整数型			

双击“时钟1”，在“\_时钟1\_周期事件”子程序中输入代码：





子程序名	返回值类型	公开	备注
_时钟1_周期事件			

变量名	类型	静态	数组	备注
窗口的句柄	整数型			
标题内容	文本型			

```
_启动窗口.移动 (取鼠标水平位置 () + 20, 取鼠标垂直位置 () + 20, 290, 80)
窗口的句柄 = 取鼠标所在窗口句柄 (取鼠标水平位置 (), 取鼠标垂直位置 ())
标题内容 = 取空白文本 (256)
-- 如果真 (窗口的句柄 ≠ 0)
  取鼠标所在窗口标题 (窗口的句柄, 标题内容, 256)
  标签1.标题 = 标题内容
```

按F5键运行程序，然后移动鼠标到QQ窗口上，可以看到“标签”中显示鼠标指向的窗口的标题，如图8-6所示。

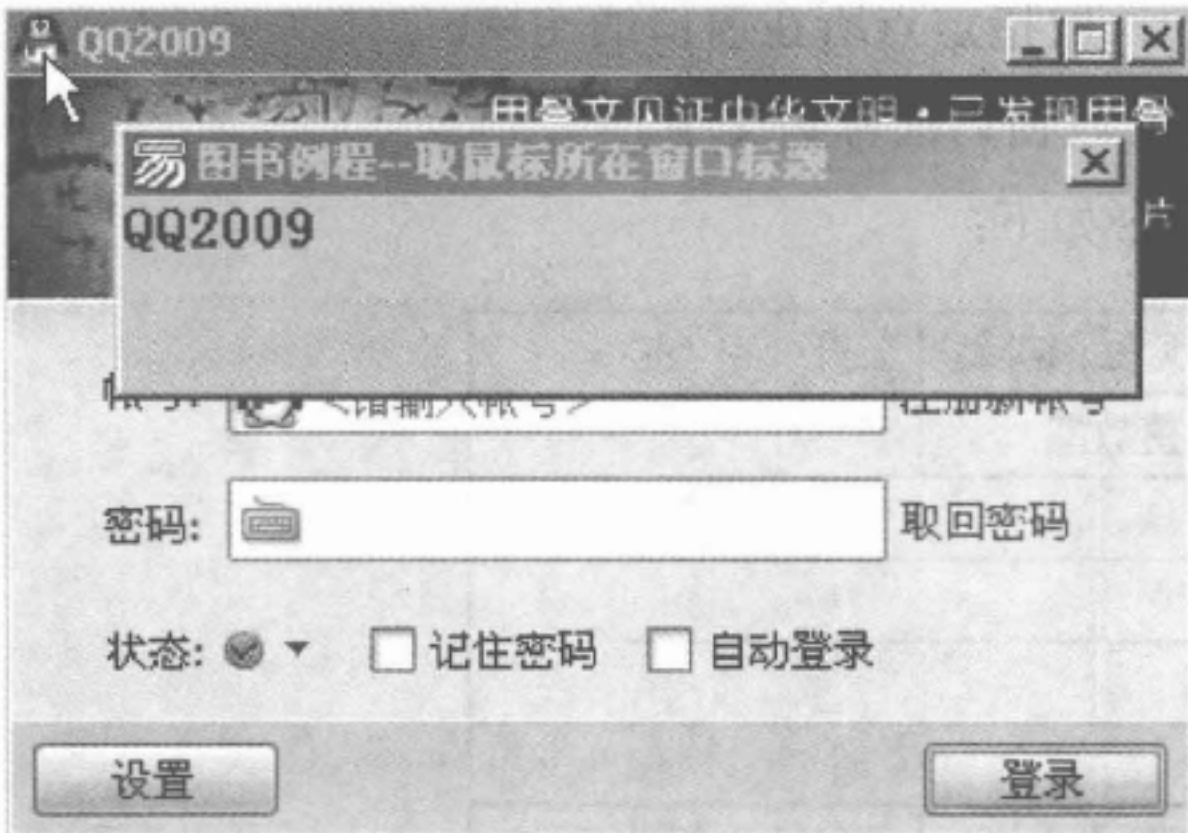


图8-6 取鼠标所在窗口标题

将鼠标移动到其他窗口上，看一看标签中显示的内容。

**【例】** 了解一下参数的数据类型是结构型的API命令。

第一步：新建一个易程序，在“\_启动窗口”中添加标签组件和时钟组件，修改其对应的名称、内容，如图8-7所示。

设置“时钟”的“时钟周期”属性为10，即每10毫秒运行一次取窗口范围的代码；“\_启动窗口”的“边框”属性设置为普通可调边框。

第二步：新建DLL命令，添加API命令相关参数。

GetWindowRect：取窗口范围矩形。

API的帮助说明，如图8-8所示。

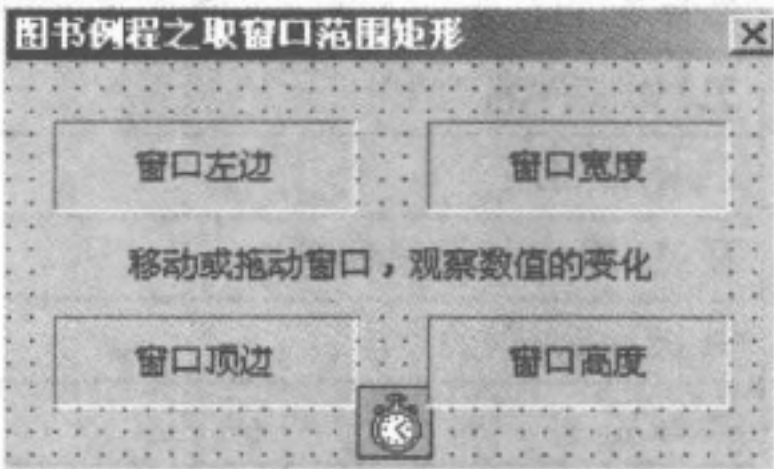


图8-7 取窗口范围矩形





GetWindowRect	
VB声明	
Declare Function GetWindowRect Lib "user32" Alias "GetWindowRect" (ByVal hwnd As Long, lpRect As RECT) As Long	
说明	
获得整个窗口的范围矩形，窗口的边框、标题栏、滚动条及菜单等都在这个矩形内	
返回值	
Long，非零表示成功，零表示失败。会设置GetLastError	
参数表	
参数	类型及说明
hwnd	Long，想获得范围矩形的那个窗口的句柄
lpRect	RECT，屏幕坐标中随同窗口装载的矩形
注解	
如将它与通过GetDesktopWindow获得的句柄联合使用，可获得对整个可视显示区域（桌面）进行说明的矩形	

图8-8 GetWindowRect帮助说明

“GetWindowRect”的DLL命令定义如下：

Dll命令名	返回值类型	公开	备 注	
取窗口矩形	整数型			
Dll库文件名：				
user32				
在Dll库中对应命令名：				
GetWindowRect				
参数名	类 型	传址	数组	备 注
窗口句柄	整数型			
矩形区域	矩形			

“GetWindowRect”命令的第二个参数的类型定义如下：

数据类型名	公开	备注		
矩形		RECT		
成员名	类型	传址	数组	备注
左边	整数型			Left
顶边	整数型			Top
右边	整数型			Right
底边	整数型			Bottom

双击“时钟1”，在“\_时钟1\_周期事件”子程序中输入代码：

子程序名	返回值类型	公开	备注	
_时钟1_周期事件				
变量名	类型	静态	数组	备注
矩形变量	矩形			
取窗口矩形（_启动窗口.取窗口句柄 0，矩形变量）				
左边标签.标题 = “左边:” + 到文本（矩形变量.左边）				
顶边标签.标题 = “顶边:” + 到文本（矩形变量.顶边）				
宽度标签.标题 = “宽度:” + 到文本（矩形变量.右边 - 矩形变量.左边）				
高度标签.标题 = “高度:” + 到文本（矩形变量.底边 - 矩形变量.顶边）				





按F5键运行程序。移动窗口或拖动窗口边框，可以看到“标签”中显示的窗口的范围变化，如图8-9所示。

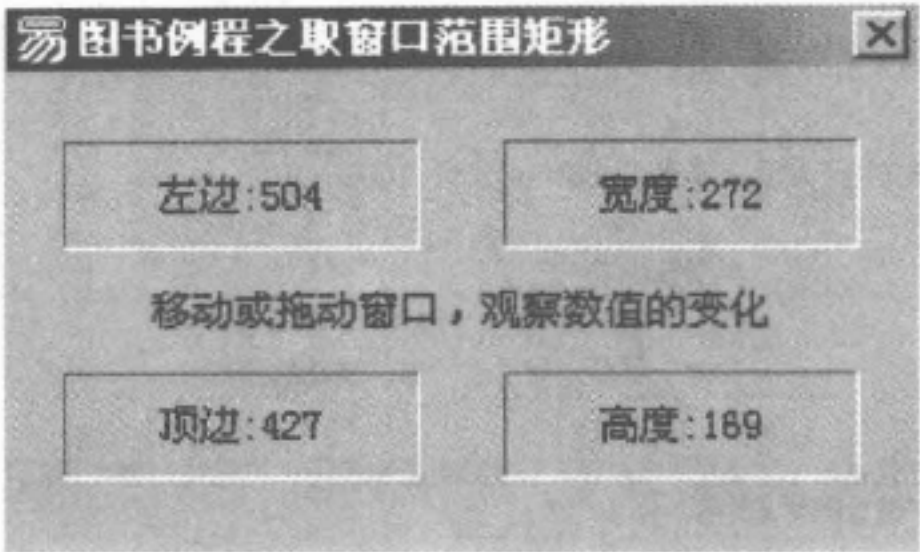


图8-9 运行后窗口范围矩形效果

8.3.2 调用非系统API

易语言还可以调用非系统自带的DLL库中的API命令。比如易语言或其他编程语言编写的DLL库中的API命令。

**【例】** 调用一个外部DLL库中的API命令，来实现在电脑屏幕上写字。  
第一步：新建一个易程序，在“\_启动窗口”中添加两个按钮，修改相应名称、标题，如图8-10所示。

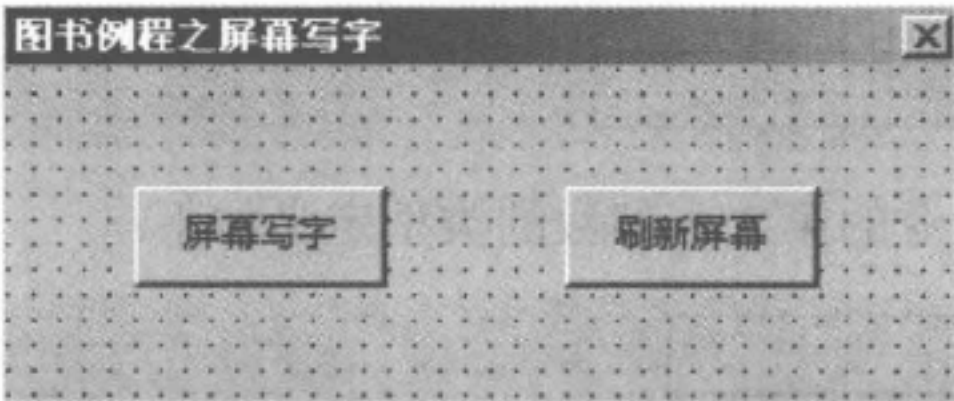


图8-10 “屏幕写字”程序界面

第二步：新建DLL命令，添加API命令相关定义。  
本例还要用到系统DLL库user32.dll中的InvalidateRect命令。该API命令屏蔽一个窗口客户区的全部或部分区域，这会导致窗口在事件期间部分重画。

Dll命令名	返回值类型	公开	备 注	
重画窗口客户区	整数型			
Dll库文件名：				
user32				
在Dll库中对应命令名：				
InvalidateRect				
参数名	类 型	传址	数组	备 注
窗口句柄	整数型			
屏蔽矩形	整数型			
重画前删除	整数型			



屏幕写字命令，根据坐标在屏幕上写字，字体的属性可以设置。（屏幕学字.dll文件可在光盘中查看）

Dll命令名	返回值类型	公开	备 注	
屏幕写字				
Dll库文件名：				
屏幕写字.dll				
在Dll库中对应命令名：				
屏幕写字				
参数名	类 型	传址	数组	备 注
横坐标	整数型			
纵坐标	整数型			
内容	文本型			
字体名称	文本型			
字体大小	整数型			
颜色	整数型			
加粗	逻辑型			
倾斜	逻辑型			
下划线	逻辑型			

双击按钮，输入如下代码：

子程序名	返回值类型	公开	备 注
_写字按钮_被单击			
屏幕写字 (20, 100, “易语言”, “黑体”, 120, #红色, 真, 真, 真)			
子程序名	返回值类型	公开	备 注
_刷新按钮_被单击			
重画窗口客户区 (0, 0, 1)			

按F5键运行程序，在桌面上就可以看到写的文字了，如图8-11所示。

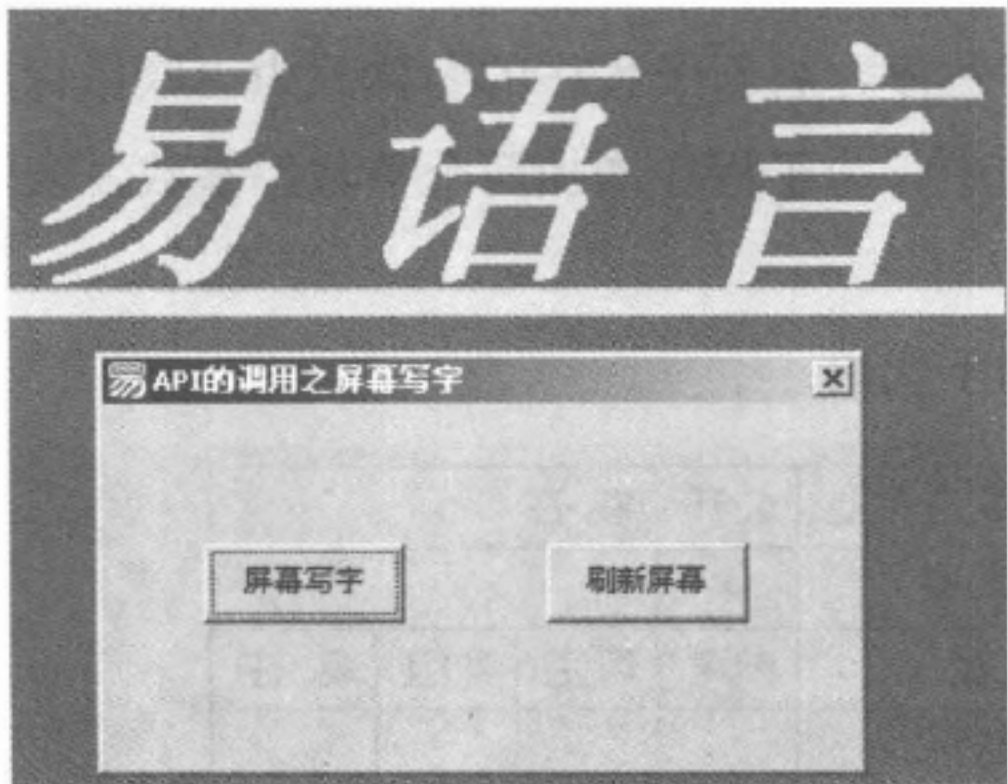


图8-11 桌面上写的文字效果





## 8.4 应用实例

前面学过了一个“取鼠标所在窗口标题”的API程序，下面用易语言来编写一个DLL程序，把这个程序中的API命令应用到要编写的DLL程序中，将其封装成一个API命令。

第一步：新建一个“Windows动态链接库”程序，先声明要调用的API命令。

Dll命令名	返回值类型	公开	备 注		
取鼠标所在窗口句柄	整数型				
Dll库文件名：					
user32.dll					
在Dll库中对应命令名：					
WindowFromPoint					
参数名	类 型	传址	数组	备 注	
横坐标	整数型				
纵坐标	整数型				

Dll命令名	返回值类型	公开	备 注		
取鼠标所在窗口标题	整数型				
Dll库文件名：					
user32.dll					
在Dll库中对应命令名：					
GetWindowTextA					
参数名	类 型	传址	数组	备 注	
窗口句柄	整数型				
内存空间	文本型	✓			
内存空间大小	整数型				

第二步：新建一个子程序，将新子程序的名称改为“取鼠标所在的窗口标题”，并将子程序的“公开”属性选中。为子程序增加两个整数型参数，表示鼠标的位置。设置子程序返回值类型为文本型。

在此子程序中输入如下代码：

子程序名	返回值类型	公开	备注		
取鼠标所在的窗口标题	文本型	✓			
参数名	类型	参考	可空	数组	备注
鼠标横坐标	整数型				
鼠标纵坐标	整数型				



变量名	类 型	静态	数组	备 注
窗口的句柄	整数型			
标题内容	文本型			

窗口的句柄 = 取鼠标所在窗口句柄 (鼠标横坐标, 鼠标纵坐标)

标题内容 = 取空白文本 (256)

--- 如果真 (窗口的句柄 ≠ 0)

取鼠标所在窗口标题 (窗口的句柄, 标题内容, 256)

返回 (标题内容)

第三步：选择菜单“编译”→“编译”项，对此程序进行编译。保存为“取鼠标所在的窗口标题.DLL”。至此，DLL程序编写完成了。下面来调用此DLL程序中的API命令。

第四步：新建一个易程序，调用自定义的DLL，来实现取鼠标所在的窗口标题的功能。

第五步：在“\_启动窗口”中添加一个标签组件和一个时钟组件。定义DLL命令。

Dll命令名	返回值类型	公开	备 注	
取鼠标所在的窗口标题	整数型			
Dll库文件名：				
取鼠标所在的窗口标题.dll				
在Dll库中对应命令名：				
取鼠标所在的窗口标题				
参数名	类 型	传址	数组	备 注
横坐标	整数型			
纵坐标	整数型			

**注解：**DLL命令原码中返回值类型是文本型，在调用时必须定义为整数型。这是因为：DLL命令运行后，结果文本保存在一段内存地址中，返回的整数型数值就是结果文本所在的内存地址的指针。DLL程序不能直接返回文本型数据和字节集型数据，当程序传递的是文本型或者字节集型，易语言将自动转换成指针。可以使用“指针到文本”或“指针到字节集”命令把结果数据从内存地址中取出来。

(1) 指针到文本命令：返回指定内存指针所指向地址处的文本，注意调用本命令前一定要确保所提供的内存指针真实有效，且指向一个以“0”字符结束的文本串。本命令的最佳使用场合就是在易语言回调子程序和易语言DLL公开子程序用作获取外部数据。

(2) 指针到字节集命令：返回指定内存指针所指向地址处的字节集，注意调用本命令前一定要确保所提供的内存指针真实有效，且保证所取数据长度不超过实际数据长度。本命令的最佳使用场合就是在易语言回调子程序和易语言DLL公开子程序用作获取外部数据。





第六步：在“\_时钟1\_周期事件”事件子程序，输入以下程序代码：

子程序名	返回值类型	公开	备注
_时钟1_周期事件			

变量名	类型	静态	数组	备注
标题内容指针	整数型			

标题内容指针 = 取鼠标所在的窗口标题 (取鼠标水平位置 (), 取鼠标垂直位置 ())

标签1.标题 = 指针到文本 (标题内容指针)

按F5键试运行一下，看看运行的结果（图8-12）。

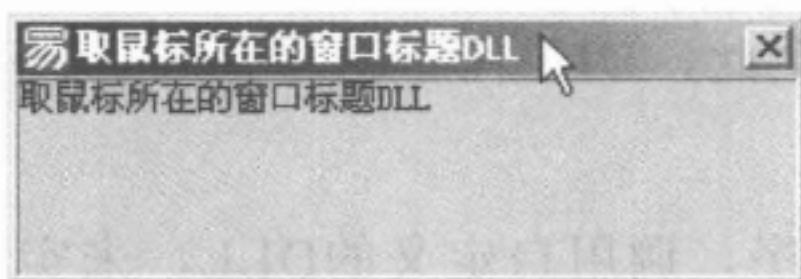


图8-12 调用的结果

在VB中同样可以调用：

新建一个VB工程，在窗口添加一个标签组件和一个时钟组件，填写如下代码，如图8-13所示。

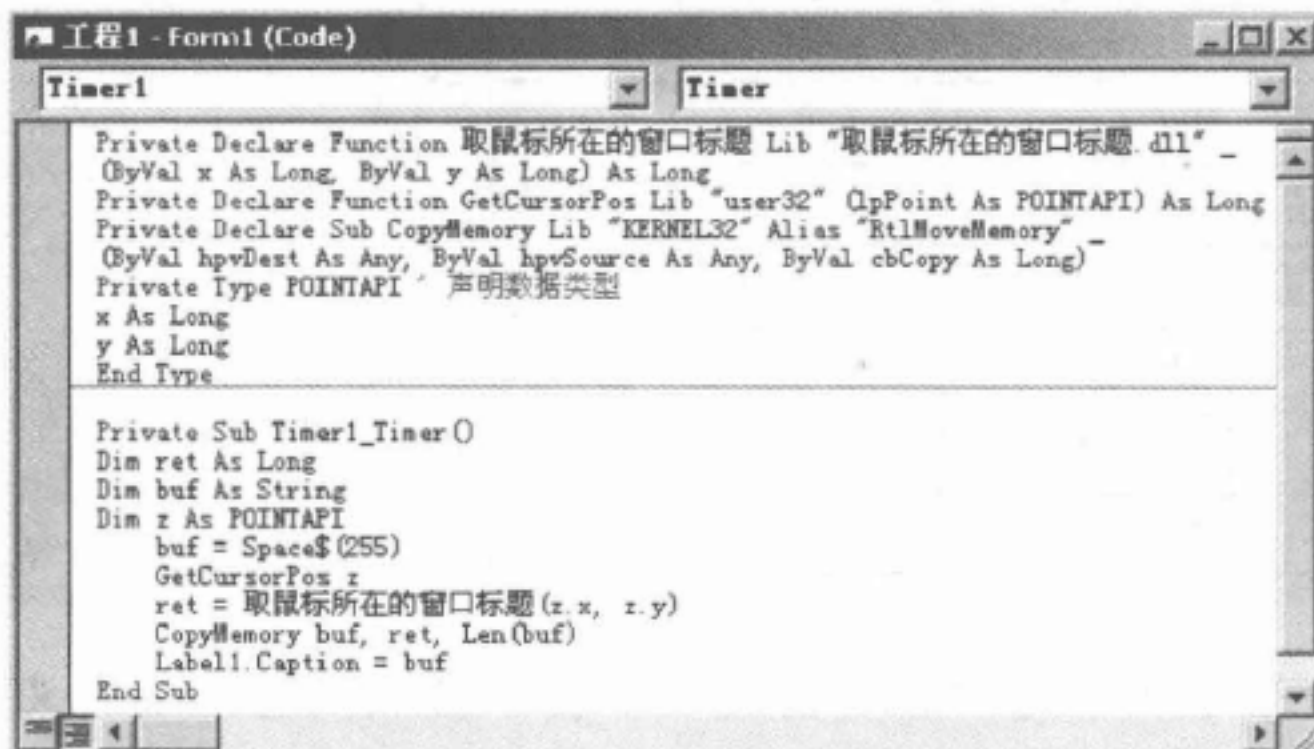


图8-13 在VB中调用DLL的代码

其中，声明了三个DLL，除易语言提供的DLL外，CopyMemory命令是用于从DLL传送给过去的内存地址中取得内容（等同于“指针到文本”命令）。GetCursorPos命令是取鼠标在屏幕中的坐标。运行后的结果如图8-14所示。



图8-14 在VB中调用DLL的运行效果

易语言编写的DLL文件、封装的API命令，符合Windows应用程序编程接口的标准，可以被其他编程语言调用；同时，易语言也可以使用其他编程语言开发的标准DLL文件，调用其封装的API命令，实现与其他编程语言的互通交流。



## 8.5 小结

简单地说，API就是一个外部命令，可以通过调用它实现一些功能。API命令来自于系统的提供和程序员使用编程语言开发。有的API参数类型有多个定义方式，这就需要在定义时根据API的功能来确定。定义API命令、API参数的数据类型与易语言的数据类型不一致时，要转换成相兼容的易语言的数据类型。互联网上有很多关于API的中文说明帮助文件，大家可以在互联网上搜索。

## 8.6 习题

- 8-1 API是\_\_\_\_\_。
- 8-2 在易语言里为了便于编程人员理解，把调用的API命令称之为\_\_\_\_\_。
- 8-3 API命令一般都在\_\_\_\_\_文件里。
- 8-4 一般地，一个窗口都具有最大化和最小化的功能，但是如果一个窗口没有上述功能怎么办？试学习编写“最小化窗口”例程（涉及到的API：CloseWindow）。
- 8-5 通常的，当鼠标指针位于一个窗口的标题栏上方时，按下鼠标左键并拖动，就可以使该窗口在屏幕上四处移动。如果要鼠标不在标题栏上方的情况下也可以四处移动窗口，那该怎么做。试学习编写“发送消息拖动窗口”例程（涉及到的API：SendMessageA）。
- 8-6 使用API命令，制作限制鼠标移动的程序（涉及到的API：ClipCursor）。
- 8-7 枚举系统中所有窗口（涉及到的API：EnumWindows，GetWindowTextA）。
- 8-8 在窗口上画五角星（涉及到的API：SetPolyFillMode，GetPolyFillMode，GetDC，Polygon）。





## 第九章 OCX组件与类型库

### 本章目标

在本章结束时，我们能够：

- 了解什么是OCX组件
- 了解什么是类型库
- 学会注册安装OCX组件
- 学会汉化和使用OCX组件
- 学会注册安装类型库
- 学会汉化和使用类型库



## 9.1 OCX组件

ActiveX（也称为OCX组件）为对象链接与嵌入定制组件，是一种可以在Windows的应用软件中使用的特殊用途的组件，是目前极为通用的Windows组件格式，组件的本质是微软公司的对象链接和嵌入（OLE）标准。由于它充分利用了面向对象的优点，使得程序效率得到了很大的提高，从而得到了广泛的应用。易语言从3.2版本开始支持直接在程序中使用此类型的组件。

OCX是对象类别扩充组件，它的名字来源于它的文件扩展名“.ocx”。控件的最早形式是以.VBX的格式出现的，后来变成了.OCX。易语言中的编辑框、按钮、图片框等都属于组件。每个组件都有自己的事件、命令和属性。组件的应用使编程非常容易；在程序的设计阶段可以设置一些属性，如大小、位置、标题等；在程序运行阶段，可以更改这些属性，还可以针对不同的事件，调用不同的命令来实现对该组件的控制。组件就好像一块块的积木，程序要做的事只是将这些积木搭起来。组件的最大好处是可以重复使用，甚至可以在不同的编程语言之间使用。

### 9.1.1 OCX组件的安装

通过“XPForm.ocx”组件，来看一下易语言中OCX组件的安装和使用。

第一步：选择易语言菜单“工具”→“类型库或OCX组件->支持库”子项，打开OCX组件或类型库包装对话框，如图9-1所示。

第二步：如果系统中没有安装“XPForm.ocx”，可先注册光盘中提供的“XPForm.ocx”文件。将“XPForm.ocx”文件保存到本地磁盘，点击“注册组件”按钮，选中要注册的OCX组件“XPForm.ocx”文件，点击“打开”按钮，组件就被注册成功了，并被添加到组件列表中，如图9-2所示。



图9-1 OCX组件或类型库包装对话框



图9-2 选择OCX组件





第三步：如果系统中已经注册了“XPForm.ocx”组件，将跳过第二步的注册组件步骤，可以直接选择要使用的组件，如图9-3所示。

第四步：选择“XP\_Form.XPForm”组件，点击“下一步”按钮，选择窗口中的OCX组件后，点击“保存”按钮可以保存成npk文件，如图9-4所示。

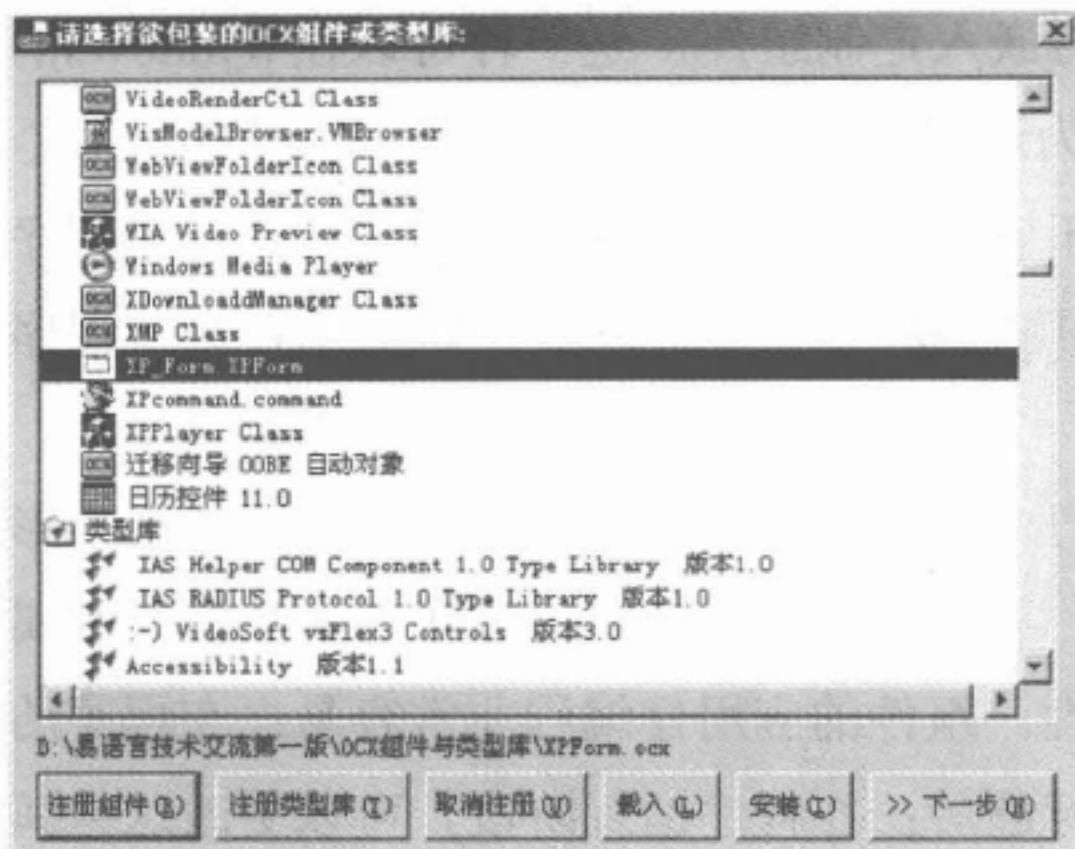


图9-3 选择要使用的OCX组件

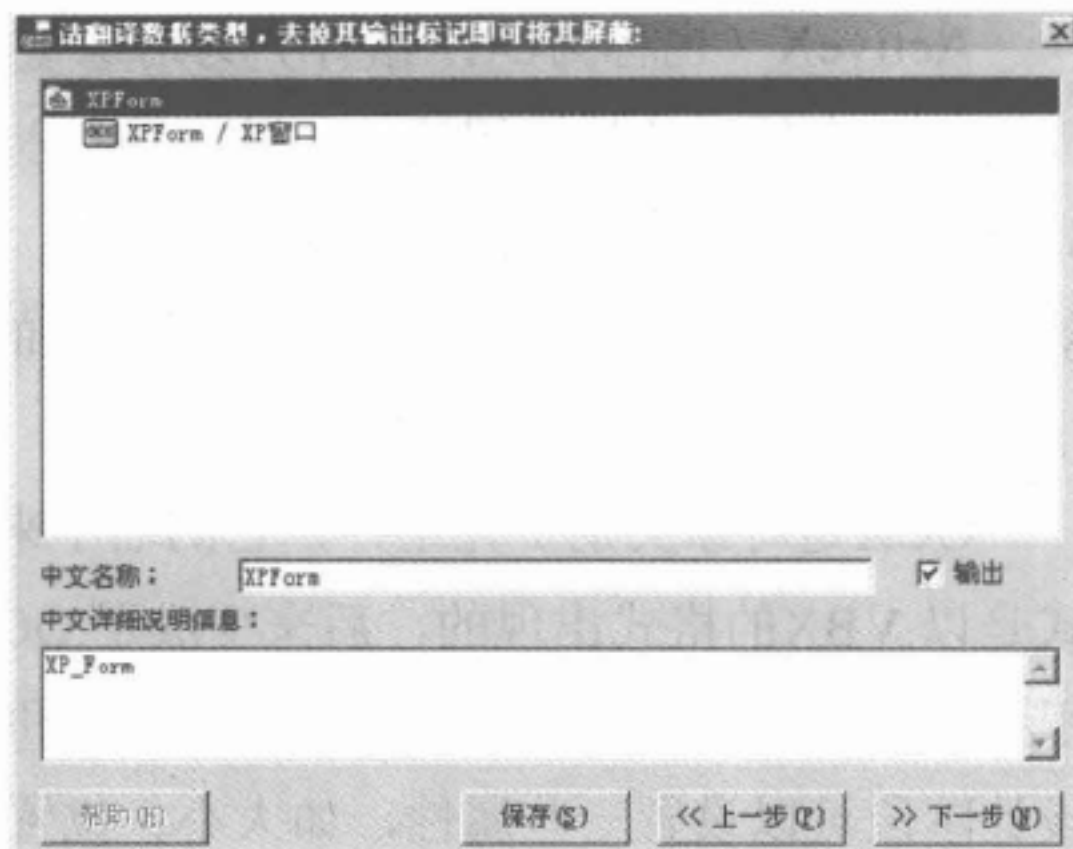


图9-4 保存注册信息

**注解：**图9-4中的下半部分，允许用户设置该类型库的中文信息（中文名称、中文详细说明信息），以及是否“输出”，如果不选择“输出”，易语言将不对其进行封装。

组件注册信息文件默认后缀为“.npk”。npk在易语言3.6版本之前名为“opk”，在3.7版本之后才更名为npk，但所有以前的opk版本在3.7之后的版本仍然支持。

npk是一个让易语言链接OCX组件的“组件包装支持库”，是一个纯文本文件，文件内容是OCX组件的接口信息及对照翻译的中文信息，易语言用这个文本文件控制显示OCX组件的文本信息，当然可以使用文本编辑器打开后根据里面的说明信息直接修改这个npk文件的内容。

本书随书光盘中提供了“XPForm.ocx”组件汉化的“XPForm.npk”文件。返回上一步，点击“安装”按钮，选中npk文件，按照提示完成安装。或者直接将“XPForm.npk”文件复制到易语言安装目录的“lib”文件夹中。就可以直接使用“XPForm.ocx”组件了，而不需要进行下面的汉化翻译步骤了。

第五步：易语言提供了汉化的功能，可以将OCX组件中所有的英文信息都翻译成为中文使用，非常方便。

**注解：**如果没有npk文件，可以按第四步方法新建npk文件，如果npk文件的内容需要修改，可在第三步点击“载入”按钮，载入npk文件。

汉化时，首先将支持库名称汉化，然后选择一个欲汉化的数据类型，点击“下一步”按钮继续汉化，如图9-5所示。

OCX组件或类型库包装对话框会列出OCX组件中欲汉化的数据类型所有的属性、命





令、事件，可以逐条汉化。如果命令或事件中含有参数的话，点击“参数”按钮对参数进行汉化，如图9-6所示。

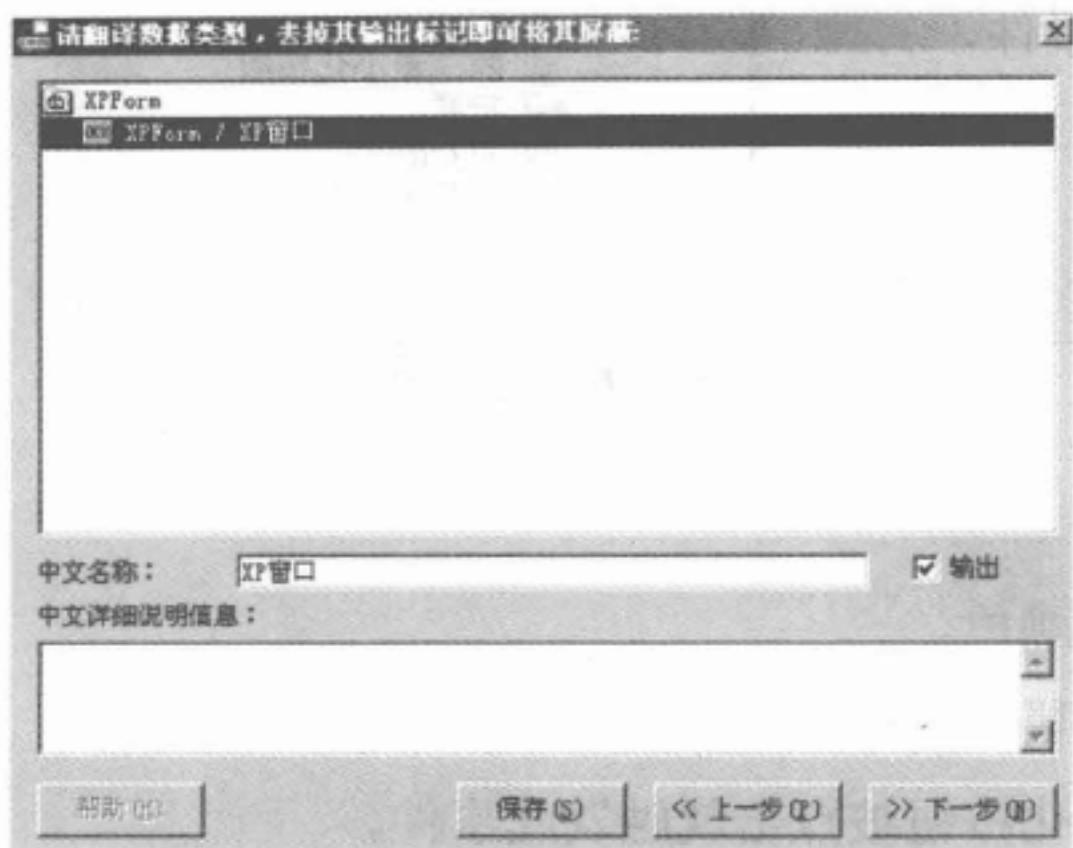


图9-5 选择支持库名称进行汉化

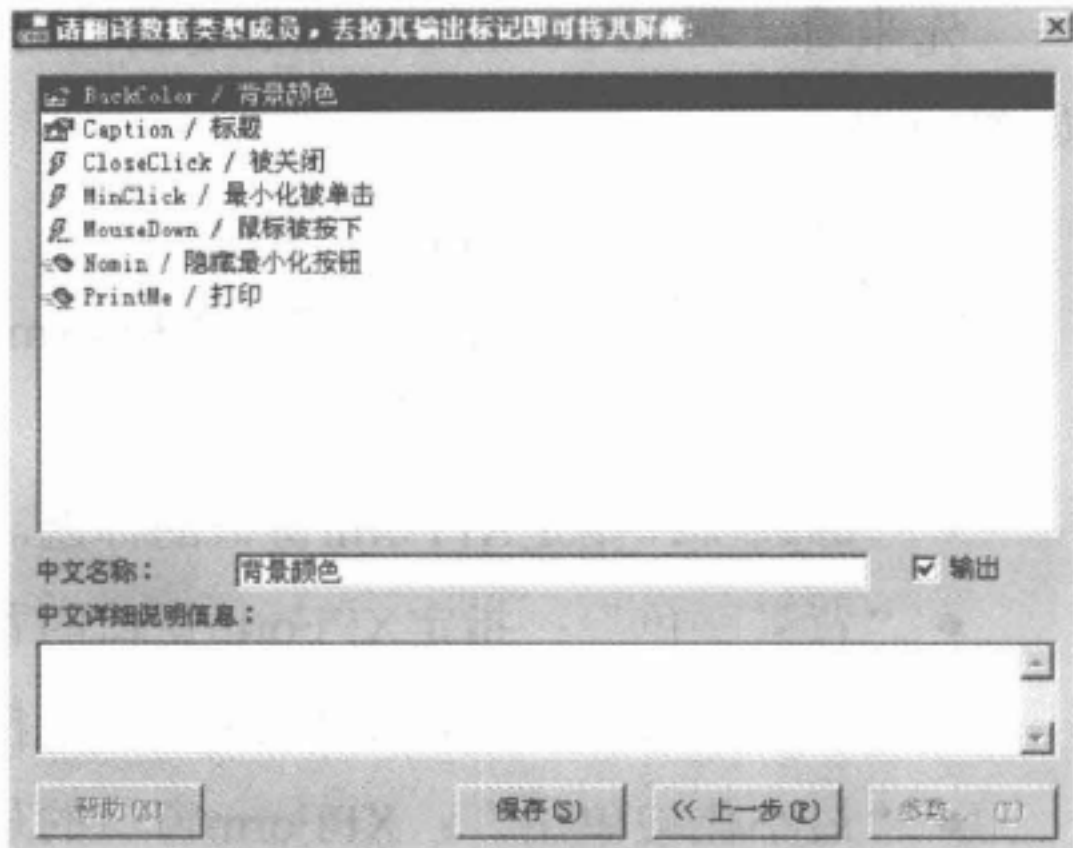


图9-6 逐条汉化组件的属性、命令和事件名称

第六步：汉化完成，点击“保存”按钮，将文件保存为npk文件。保存npk文件成功会弹出信息框提示npk文件已创建成功。并且按照信息框的提示将npk文件载入到易语言系统中使用。

第七步：重新启动易语言，通过易语言的菜单“工具”→“支持库配置”，打开支持库设置对话框。选中“XPForm1.0版”，如图9-7所示。

第八步：点击“确认”按钮，这样就会在工具箱的外部组件面板中看到一个新的OCX组件按钮了，如图9-8所示。

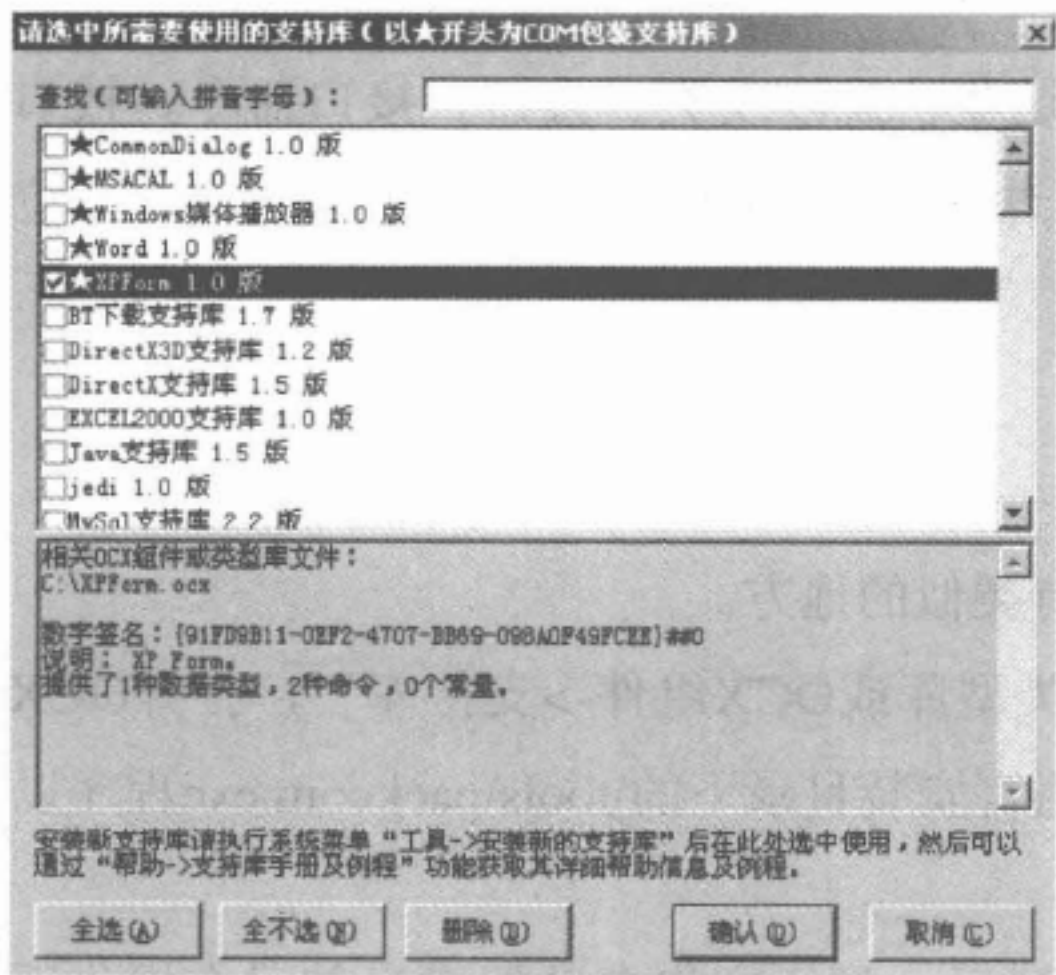


图9-7 支持库配置XPForm支持库



图9-8 新的组件

### 9.1.2 OCX组件的使用

在OCX组件安装与汉化完成之后，就可以对其进行使用了，下面就一起来看看这个





OCX组件的使用。用XPForm组件，来做一个模仿XP窗口样式的窗口程序。

先来看一看XPForm组件的主要属性命令事件，如图9-9所示。

- “打印”：打印XPForm窗口。
- “隐藏最小化按钮”：隐藏XPForm窗口的最小化按钮。
- “标题”：指定XPForm窗口的标题。
- “背景颜色”：指定XPForm窗口的背景颜色。
- “鼠标被按下”：在XPForm窗口上按下鼠标就触发此事件。
- “最小化被单击”：XPForm窗口的最小化按钮被点击就触发此事件。
- “被关闭”：XPForm窗口的关闭按钮被点击就触发此事件。

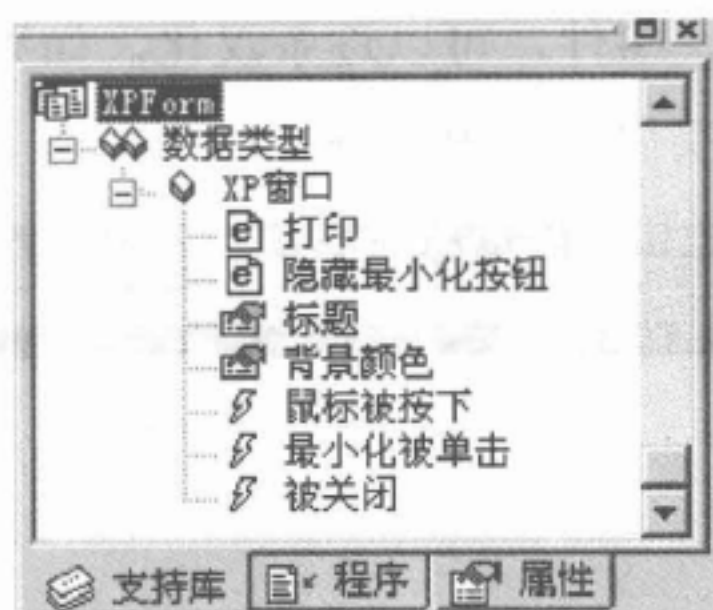


图9-9 XPForm组件的属性命令事件

新建一个Windows窗口程序，在启动窗口中添加组件“XP窗口1”，在“\_启动窗口\_创建完毕”事件中编写如下代码：

```
XP窗口1.移动(0,0,_启动窗口.宽度,_启动窗口.高度)
```

按F5键运行程序，这就是一个简单的模仿XP风格窗口的OCX组件。

## 9.2 类型库

类型库是许多类型信息的集合，这些信息涉及类、接口、接口命令、接口命令的参数及相关的常量和事件等。类型库通过ITYPELIB接口来访问，类型库通常位于\*.tlb，\*.olb，\*.dll，\*.ocx，\*.sys等文件中。

### 9.2.1 类型库的安装

类型库的安装与OCX组件的安装有类似的地方。

第一步：选择菜单“工具”→“类型库或OCX组件->支持库”，打开OCX组件或类型库包装对话框；也可以直接运行易语言安装目录下的\tools\packcom.exe程序以启动OCX组件或类型库包装对话框。

OCX组件或类型库包装对话框打开后，易语言将自动搜索当前已在操作系统中注册的类型库（见图9-10）及在OCX组件后显示出所有列表。整个列表分为两大类：“OCX组件”和“类型库”。没有注册类型库的，要首先注册类型库。注册类型库的步骤与注册OCX组件是一样的。

第二步：选择一个已注册过的类型库“Microsoft Word 11.0 Object Library 版本8.3”，





这是一个WORD类型库，也就是说，当前系统中要事先安装WORD程序，才能找到并使用该类型库。已经注册过的类型库，可以载入npk文件进行修改，或直接安装npk文件使用。点击“>>下一步”按钮对类型库进行汉化。

此时，会看到一个类似注册OCX的窗口，在这里可以对类型库作一些汉化等操作。在完成了所需要的汉化工作之后，点击“保存”即可，如图9-11所示。

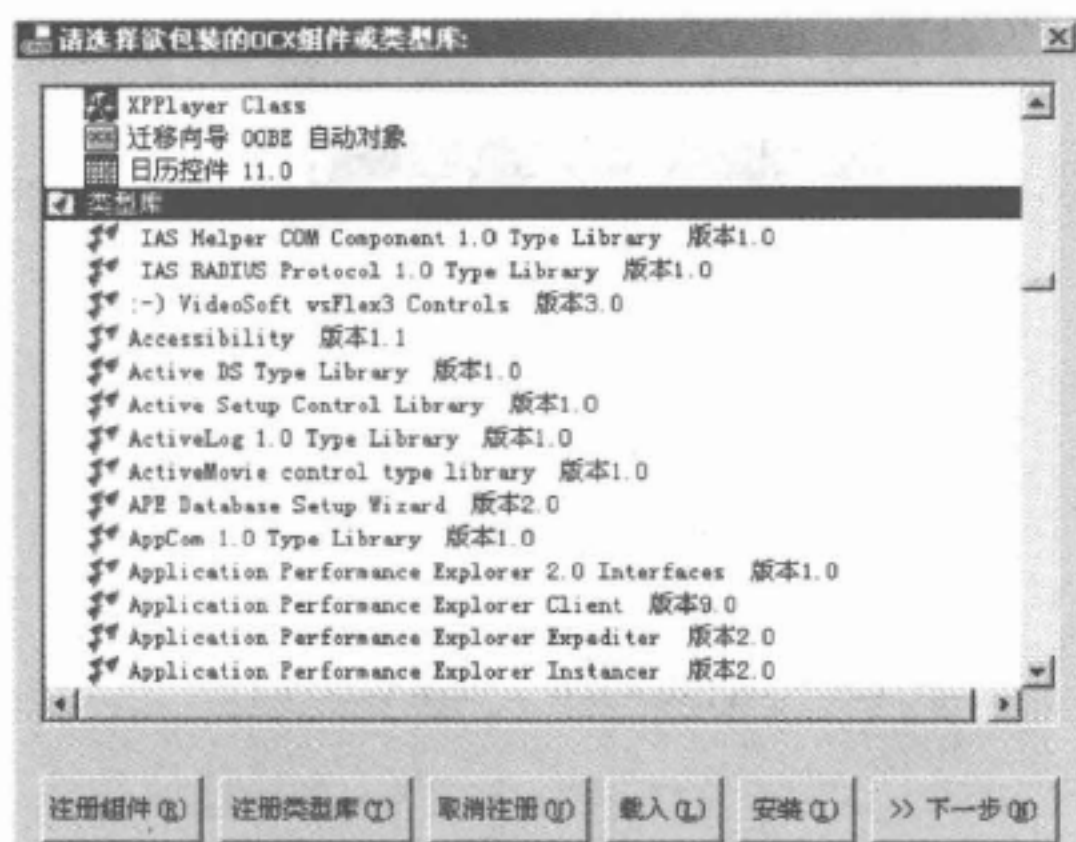


图9-10 已注册的类型库



图9-11 汉化WORD类型库

第三步：汉化完成，将文件保存为npk文件。保存npk文件成功会弹出信息框提示npk文件已创建成功。并且按照信息框的提示将npk文件载入到易语言系统中使用。

第四步：重新启动易语言，通过易语言的菜单“工具”→“支持库配置”，从而打开支持库配置对话框。选中“Word1.0版”，如图9-12所示。

安装的Word支持库数据类型列表如图9-13所示，包括Word的各子数据类型及其命令、属性、事件。

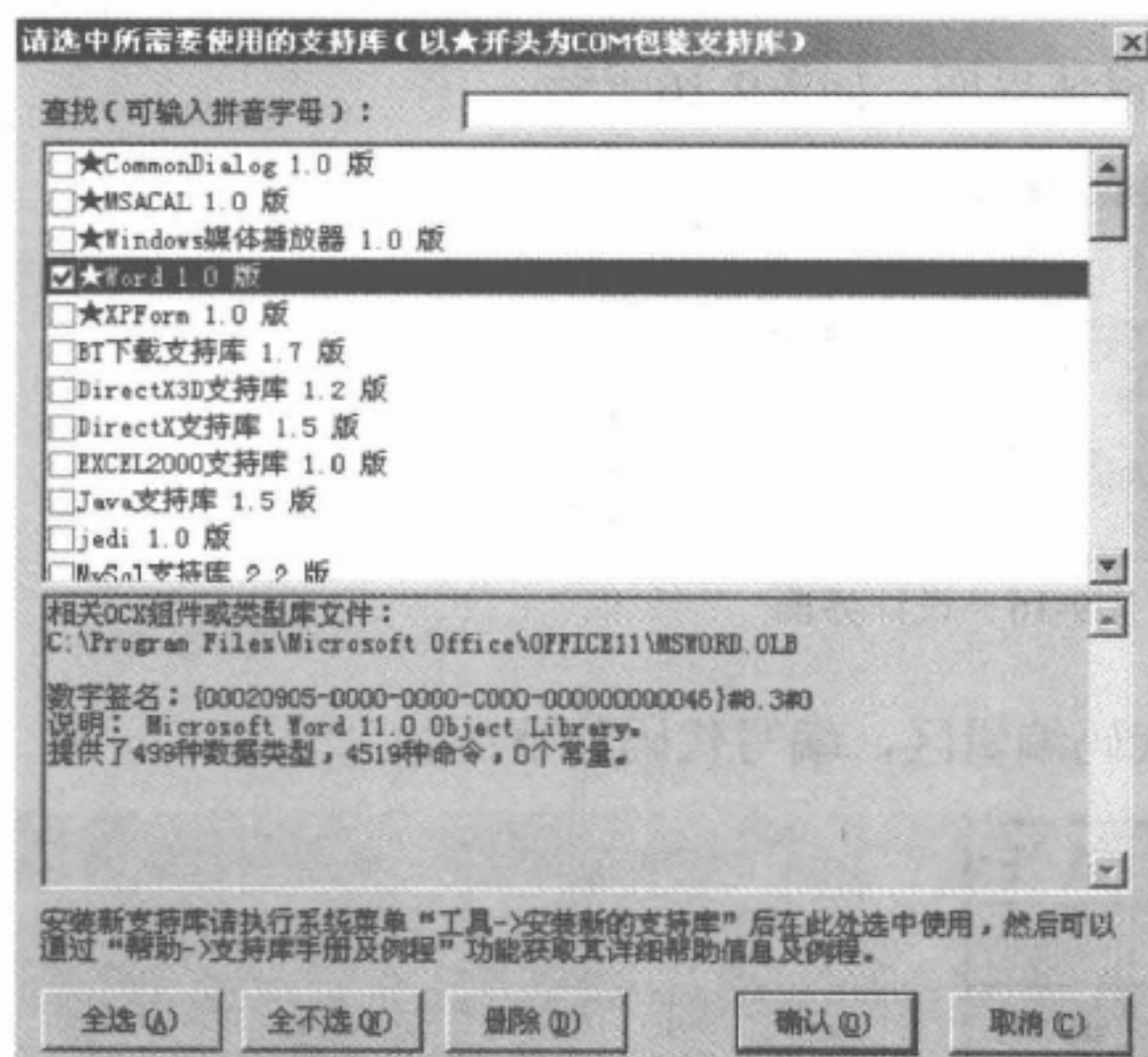


图9-12 支持库配置Word类型库



图9-13 Word数据类型





当安装类型库时，易语言会为每个数据类型自动增加以下命令中的0个或多个：“清除（）”、“创建（）”、“获取（）”、“是否为空（）”、“取子对象（）”、“取接口（）”、“挂接事件（）”等。至于具体增加其中的哪几个命令，则视不同的数据类型而定。如果类型库中包含“事件组件”，则安装后该事件组件图标将出现在组件箱的外部事件组件中，如图9-14所示。

“Application”对象的所有触发事件如图9-15所示。



图9-14 “Application”对象事件组件

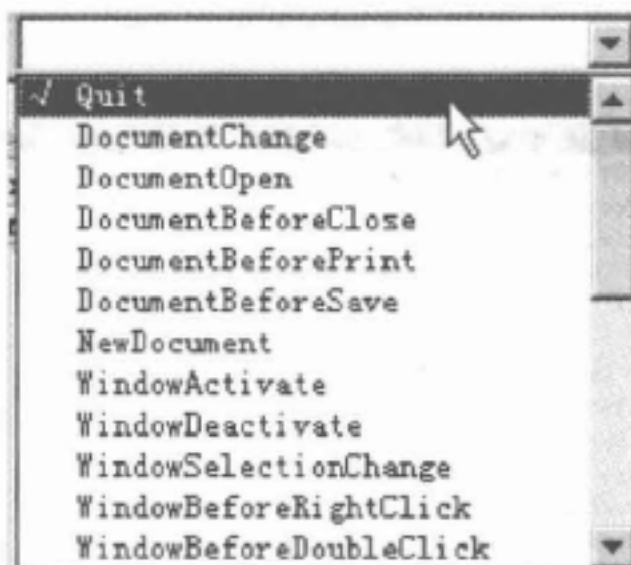


图9-15 “Application”对象触发事件

至此，类型库的安装操作全部完成。

在此重点说明其中的“挂接事件”命令：易语言会自动为拥有该“事件组件”的数据类型添加“挂接事件”命令，该命令有一个类型为“相应事件组件”的参数。调用数据类型的“挂接事件”命令可将“事件组件”（包括其中的事件处理代码）“挂接”到该数据类型上。用户只有主动调用“挂接事件”命令，自己编写的事件处理代码才会正常工作。通常可在窗口的“创建完毕”事件中调用“挂接事件”命令。

## 9.2.2 类型库的使用

为了方便了解，接下来将通过“Word类型库”来编写一个简单的例程。

第一步：新建一个易程序，设计窗体界面，如图9-16所示。

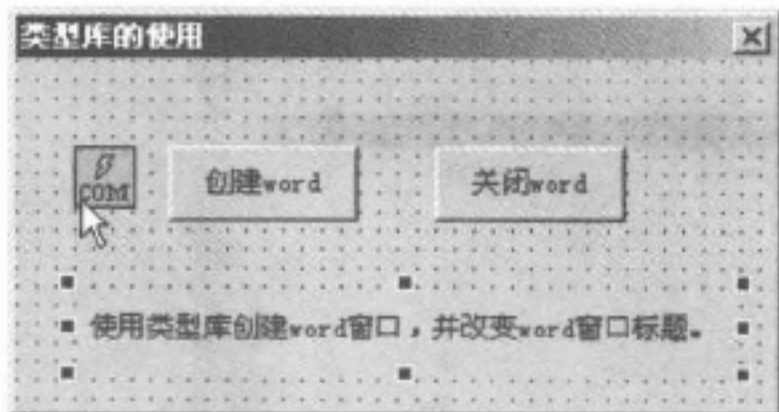


图9-16 设计界面

第二步：双击“按钮1”，进入代码编辑区，编写代码如下：

窗口程序集名	保留	保留	备注
窗口程序集1			
变量名	类型	数组	备注
word类型库	Application		





子程序名	返回值类型	公开	备注
_按钮1_被单击			

```
word类型库.Visible = 真
word类型库.Caption = “易语言word”
```

子程序名	返回值类型	公开	备注
_按钮2_被单击			

```
word类型库.Quit (, )
```

子程序名	返回值类型	公开	备注
_Application事件1_Quit			

```
信息框 (“即将关闭word”, 0, )
```

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

```
word类型库.挂接事件 (Application事件1)
```

代码中：

首先创建一个程序集变量，类型为“Application”，这样就可以直接操作“Application”中的命令。

```
word类型库.挂接事件 (Application事件1)
```

将“事件组件”（包括其中的事件处理代码）“挂接”到该数据类型上。

```
word类型库.Visible = 真
word类型库.Caption = “易语言word”
```

设置Word程序可见，并且设置Word程序标题为“易语言word”。

```
word类型库.Quit (, )
```

关闭Word程序。

```
信息框 (“即将关闭word”, 0, )
```

触发Word程序被关闭事件，弹出信息框提示。

第三步：程序编写完毕，按F5键运行程序，看下效果是否与需要的一致。

可以将OCX组件和类型库当作易语言的外部支持库来使用，扩展易语言的功能，使易语言更加强大。

### 9.3 小结

OCX组件可以包含组件，也可以包含数据类型。同样，类型库可以包含数据类型，也可以包含组件。OCX组件可以是类型库，类型库也可以是OCX组件。内容可以一样，





区别只在于接口规范的不同。易语言可以调用大量现成的OCX和类型库实现强大的功能，加快程序的编写速度。

## 9.4 习题

- 9-1 OCX是对象类别扩充组件，它的名字来源于\_\_\_\_\_。
- 9-2 类型库是\_\_\_\_\_。
- 9-3 OCX组件注册信息文件默认后缀为\_\_\_\_\_。
- 9-4 类型库通常位于\_\_\_\_、\_\_\_\_、\_\_\_\_、\_\_\_\_、\_\_\_\_等文件中。
- 9-5 按本章OCX组件和类型库的安装注册汉化步骤调用Microsoft日历组件11.0版本7.0（OCX组件可在随书光盘中找到）。







## 第十章 面向对象

### 本章目标

在本章结束时，我们能够：

- 了解什么是面向对象
- 了解什么是类
- 了解类与对象的关系
- 掌握类的特性
- 掌握使用类模块来编程





## 10.1 了解面向对象

面向对象（Object Oriented，简称OO）是当前最流行的编程方式。面向对象的概念和应用已超越了程序设计和软件开发，扩展到很宽的范围。如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD技术、人工智能等领域。

从表面来看，面向对象这个术语的意思是，把软件组织成一系列离散的、合并了数据结构和行为的对象。这与以前软件开发方法中数据结构和行为只是松散关联是不同的。谈到面向对象，这方面的文章非常多。但是，明确地给出面向对象的定义或说明面向对象定义的非常少。起初，“面向对象”是专指在程序设计中采用封装、继承、抽象等设计方法。可是，这个定义显然不再适合现在的情况。面向对象的思想已经涉及到软件开发的各个方面。如：面向对象的分析（OOA，Object Oriented Analysis）、面向对象的设计（OOD，Object Oriented Design）及常说的面向对象的编程实现（OOP，Object Oriented Programming）。许多有关面向对象的文章都只是讲述在面向对象的开发中所需要注意的问题或所采用的比较好的设计方法。看这些文章只有真正懂得什么是对象，什么是面向对象，才能最大程度地对自己有所裨益。这一点，恐怕对初学者甚至是从事相关工作多年的人员也会对它们的概念模糊不清。

易语言在全中文编程的基础上，从3.8版推出后，也开始全面支持面向对象编程。并且在中文编程的基础上，简化了类的创建与使用，使面向对象的编写过程变得更加简单。

### 10.1.1 类的概念

要了解面向对象首先要了解类的概念。而类和对象是两个截然不同的概念，所以不要将类和对象混淆。

对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象。对象不仅能表示具体的事物，还能表示抽象的规则、计划或事件。对象具有状态，一个对象用数据值来描述它的状态。对象还有操作，用于改变对象的状态，对象及其操作就是对象的行为。对象实现了数据和操作的结合，使数据和操作封装于对象的统一体中。

具有相同或相似性质的对象的抽象化就是类。因此，对象的抽象化是类，类的具体化就是对象，也可以说：类的实例是对象。

如果对类和对象的描述还比较模糊，下面就举个实例来进一步了解类与对象的概念。“飞机”是一个类，飞机有什么特点呢？飞机包括“用途”、“载重”、“颜色”等特点。“客机”属于“飞机”这个群体，它具有“飞机”这类群体的所有特点，所以说“客机”是“飞机”类的一个实例，也就是说“客机”是基于“飞机”类的一个对象。





## 10.1.2 类的创建

下面来看一下，在易语言中怎样创建一个类。

第一步：新建一个类，可以通过菜单“插入”→“类模块”来实现，如图10-1所示。

也可以通过在程序面板中的“程序数据”项目上单击鼠标右键，在弹出菜单中选择“新类模块”项，如图10-2所示。

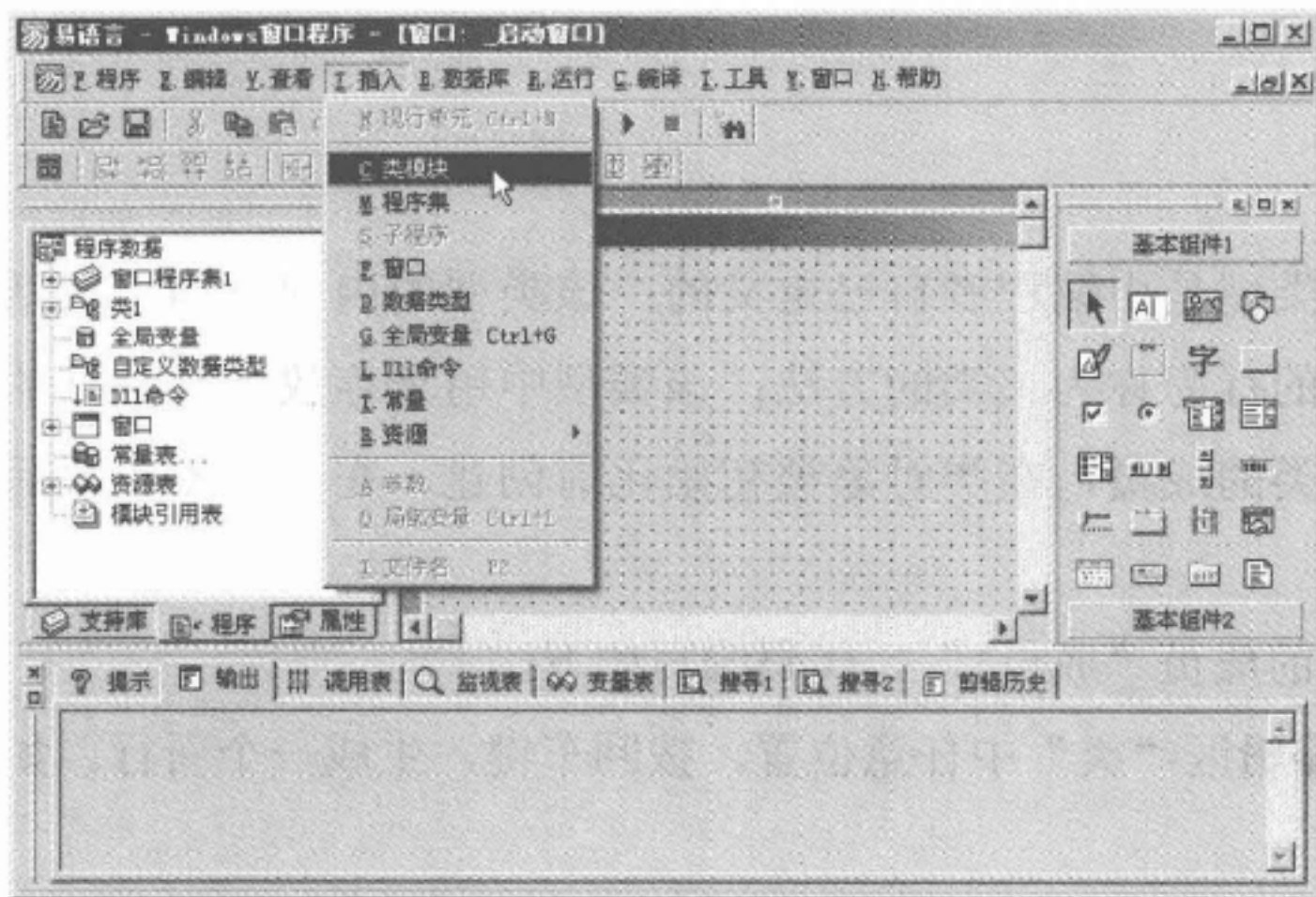


图10-1 从菜单新建类模块



图10-2 从程序面板新建类模块

第二步：插入后的类，会自动生成“\_初始化（）”方法和“\_销毁（）”方法，如图10-3所示。

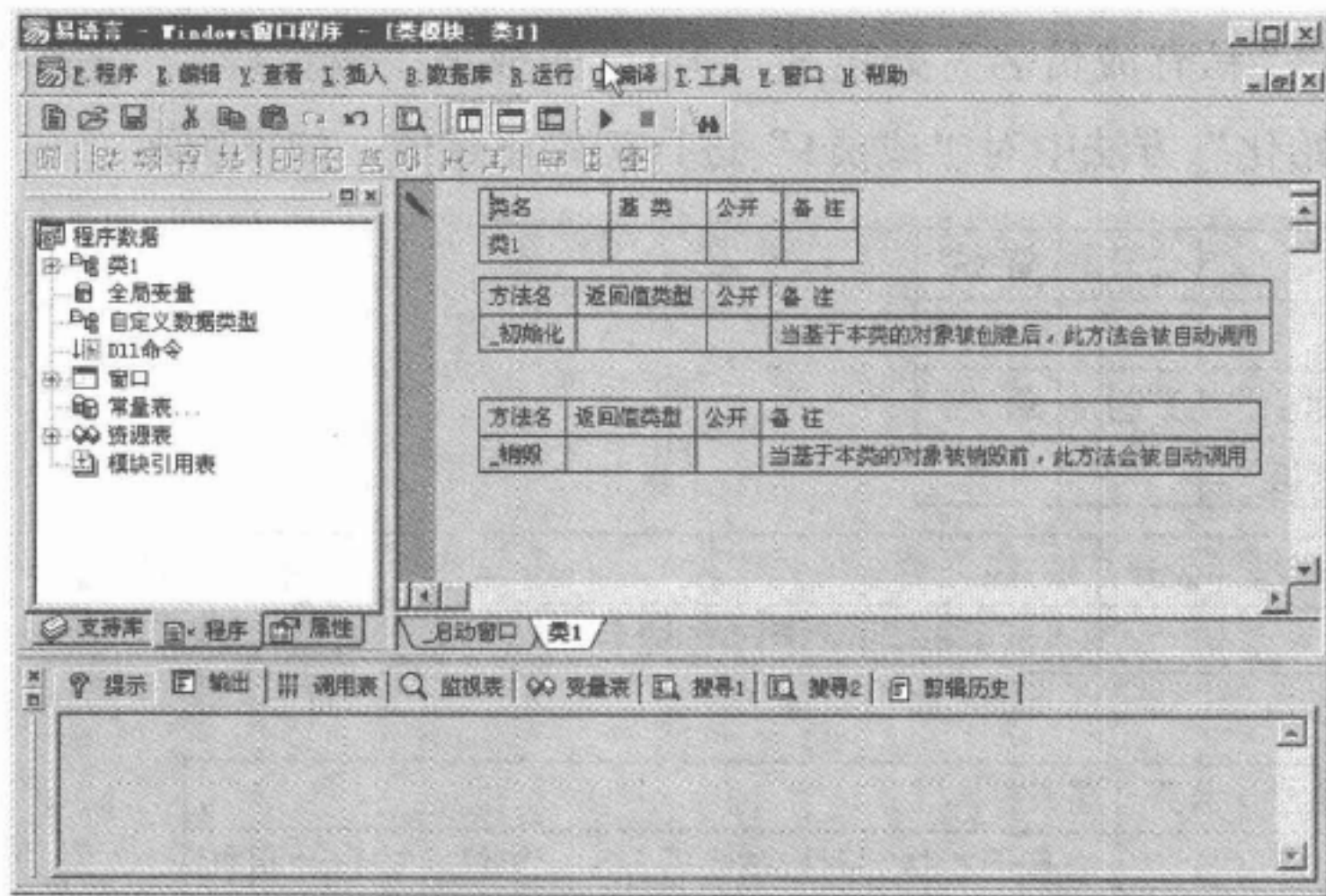


图10-3 类的创建

“\_初始化（）”方法在基于该类的对象被创建后，会被自动调用；“\_销毁（）”方法，在基于该类的对象被销毁前，会被自动调用。





在子程序中，可以编写代码，完成程序的某些设置。如：在“\_初始化（）”方法中，设置各成员的初始值；在“\_销毁（）”方法中，处理关闭文件、关闭数据库等操作。

在易语言中，类的创建和释放的过程是自动的。创建顺序为：先创建基类对象再创建其继承类对象；释放顺序为：先释放继承类对象再释放基类对象。

## 10.2 类的特性

### 10.2.1 类的封装性

类可以看做一个类型，这种类型是由编程者自己定义的；该类型的内部结构中包括该类型中的数据和行为；该类型的行为称为该类的方法；该类型中用来存放各种数据的变量，是该类的成员。易语言中类的成员，在类对象被初始化前创建，在类对象销毁后释放。

新建一个类“类1”，插入类的成员“成员1”。代码编写如下：  
第一步：将光标停留在代码编辑区“类”中任意位置，按回车键，生成一个新行。如下所示：

类名	基类	公开	备注
类1			
私有成员名	类型	数组	备注

第二步：在“私有成员名”处定义“类1”的成员“成员1”，设置数据类型为整数型。并在“\_初始化”方法中对“成员1”进行初始化赋值。如下所示：

类名	基类	公开	备注
类1			
私有成员名	类型	数组	备注
成员1	整数型		

方法名	返回值类型	公开	备注
_初始化			当基于本类的对象被创建后，此方法会被自动调用

成员1 = 1

方法名	返回值类型	公开	备注
_销毁			当基于本类的对象被销毁前，此方法会被自动调用

易语言中类的成员是私有的，只能被类自身的方法所调用，不能被其他类所调用。要想读取或赋值类的成员，就必须通过新建专门的公开方法来实现，即对象调方法，方法来操作成员。存取“类1”的“成员1”代码如下所示：



方法名	返回值类型	公开	备注
取成员1方法	整数型	✓	

返回 (成员1)

方法名	返回值类型	公开	备 注		
置成员1方法		✓			
参数名	类 型	参考	可空	数组	备 注
数值	整数型				

成员1 = 数值

那么类的方法是怎么来的呢？创建类的方法与创建易语言的子程序是一样的，在类模块的代码编辑区右键菜单选择新方法或“Ctrl+N”加入类的方法。如下所示：

方法名	返回值类型	公开	备注
方法1			

易语言中，类的方法可以设置“公开”属性。没有设置“公开”属性的方法，在基于该类的对象创建后是不可见的，这样可以达到信息隐藏的目的，即让类仅仅公开必须要让外界知道的内容，而隐藏其他一切内容。设置了“公开”属性的方法，在该类的对象被创建后就可以调用了。

下面的代码就是定义的类的“公开”方法和“不公开”方法。

方法名	返回值类型	公开	备注
方法1	整数型	✓	

信息框 (成员1, 0, )

方法名	返回值类型	公开	备注
方法2	整数型		

信息框 (2, 0, )

“公开”的方法可以被类代码外部通过该类的对象实体来调用。没有“公开”的方法，只能被该类本身和该类的子类所调用。

类的封装是一种信息隐蔽技术，它体现于类的说明，是对象的重要特性。封装使数据和操作该数据的方法封装为一个整体，以实现独立性很强的模块，使得用户只能见到对象的外特性（对象能接受哪些消息，具有哪些处理能力），而对象的内特性（保存内部状态的私有数据和实现功能的算法）对用户是隐蔽的。封装的目的在于把对象的设计者和对象的使用者分开，使用者不必知晓行为实现的细节，只需用设计者提供的方法来完成所需要的工作。

### 10.2.2 类的继承性

一个类的基类叫做该类的父类，反之，该类叫做父类的子类。





一个类既可能有子类，也可能有父类。

一个父类可以派生出许多子类，这些子类叫做其父类的派生类，也叫其父类的继承类。

继承性是子类自动共享父类的数据和方法的机制。它由类的派生功能体现。一个类直接继承其他类的全部描述，同时可修改和扩充。继承具有传递性。类的对象是各自封闭的，如果没继承性机制，则类对象中数据、方法就会出现大量重复。继承不仅支持系统的可重用性，而且还促进系统的可扩充性。

在子类中可以以“类名.方法名”的方式指定访问父类中的方法。任何类均可以指定另外一类作为其父类，继承层数不限，但不能递归嵌套既类1的父类是类2，类2的父类是类1。

前面已经定义了“类1”及其成员和方法，接下来新建一个类“类2”，将“类2”的基类定义为“类1”，那么“类2”将继承“类1”的所有公开方法，并可以直接使用基类中公开和未公开的方法。

类名	基类	公开	备注
类2	类1		

方法名	返回值类型	公开	备注
_初始化			当基于本类的对象被创建后，此方法会被自动调用

方法名	返回值类型	公开	备注
_销毁			当基于本类的对象被销毁前，此方法会被自动调用

定义一个“类2”的变量“类变量”，调用基类“类1”的方法。

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
类变量	类2			

类变量.方法1()

### 10.2.3 类的多态性

易语言中，类的多态性是指父类和子类中，都拥有定义相同（相同名称、相同参数）的方法，这些方法执行不同的操作，从而实现“一个接口，多种实现”。

第一步：新建一个类1，并为类1添加方法1，并公开它。

类名	基类	公开	备注
类1			

方法名	返回值类型	公开	备注
_初始化			当基于本类的对象被创建后，此方法会被自动调用





方法名	返回值类型	公开	备 注
_销毁			当基于本类的对象被销毁前，此方法会被自动调用

方法名	返回值类型	公开	备 注
方法1		✓	

信息框 (“类1方法1”, 0, )

第二步：再建一个类2，定义基类为类1，再加入类2的方法1。

类名	基 类	公开	备 注
类2	类1		

方法名	返回值类型	公开	备 注
_初始化			当基于本类的对象被创建后，此方法会被自动调用

方法名	返回值类型	公开	备 注
_销毁			当基于本类的对象被销毁前，此方法会被自动调用

方法名	返回值类型	公开	备 注
方法1		✓	

信息框 (“类2方法1”, 0, )

定义一个“类2”的变量“类2变量”，和一个“类1”的变量“类1变量”。将子类的“类2变量”赋值给父类的“类1变量”，然后，“类1变量”调用的“方法1”，执行的是“类2”中的代码。

**注解：**（1）所调用的方法的名称和参数必须相同。

（2）不能将父类变量赋值给子类变量。

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

变量名	类 型	静态	数组	备 注
类1变量	类1			
类2变量	类2			

类1变量 = 类2变量

类1变量.方法1 ()

运行结果，显示信息框“类2方法1”。

这种将子类变量赋值给父类变量，从而实现父类方法执行子类方法中的代码，体现了类的多态性。

在面向对象方法中，对象和传递消息分别表现事物及事物间相互联系的概念。类和继承是适应人们一般思维方式的描述。方法是允许作用于该类对象上的各种操作。这种对





象、类、消息和方法的程序设计方式的基本点在于类的封装性和类的继承性。通过封装能将对象的定义和对象的实现分开，通过继承能体现类与类之间的关系，以及由此带来的动态联系和实体的多态性，从而构成了面向对象的基本特征。

## 10.3 小结

易语言支持用户定义和使用类，支持类的创建、释放、封装、继承、多态等特性。类的创建顺序为：先创建基类对象，再创建其继承类对象，如果类中具有对象成员，则先于其所处对象创建。类的释放顺序为：先释放继承类对象，再释放基类对象，如果类中具有对象成员，则在其所处对象后释放。类的所有成员变量只能由该类本身的方法所操作，属于私有性质。所以需要通过调用类中的方法，来间接操作成员的值。

任何类均可以指定另外一类作为其基类，继承层数不限，但不能递归嵌套，即类1的父类是类2，类2的父类是类1，这样间接继承自身是不允许的。在继承类中可以以“类名.方法名”的方式指定访问基础类中的方法。只有标记为“公开”的方法才能在类代码外部通过该类的对象实体来访问。可以将一个子类对象变量赋予到其父类数据类型变量中，此时对此父类对象变量进行操作，将反映出类的多态性。

## 10.4 习题

- 10-1 对象是\_\_\_\_\_。
- 10-2 类是\_\_\_\_\_。
- 10-3 易语言中的类的成员是\_\_\_\_\_。
- 10-4 在继承类中可以以\_\_\_\_\_的方式指定访问基础类中的方法。
- 10-5 只有标记为\_\_\_\_\_的方法才能在类代码外部通过该类的对象实体来访问。
- 10-6 类都有哪些特性？
- 10-7 类与对象有什么关系？
- 10-8 类的创建顺序是什么？
- 10-9 类的释放顺序是什么？







## 第十一章 对象和COM对象

### 本章目标

在本章结束时，我们能够：

- 了解什么是对象
- 了解什么是COM对象
- 掌握使用对象
- 掌握使用对象的查看命令
- 掌握使用COM对象





## 11.1 对象的应用

所谓“对象”即为属性、命令和事件的集合。对象（Object）是一件事、一个实体、一个名词、可以获得的东西、可以想象有自己的标识的任何东西。比如，在现实生活中，电脑就是我们搜寻资料的一个对象，它具有外观、操作系统、价格等特点，这在对象概念中被称为属性；而利用这个电脑玩游戏、看电影、查找资料等应用，这就对应于对象里的命令；使用电脑过程中发生的事情，如：电脑损坏，对应于对象里的事件。对象的属性记录对象特征，对象的命令提供了对对象进行某种操作的途径，对象的事件用作通知外部，它的状态发生了改变。

易语言是一种基于对象的编程语言，能否熟练地使用易语言，理解对象的概念及其使用方法非常重要。

COM对象是遵循COM规范编写、以Win32动态链接库（DLL）或可执行文件（EXE）形式发布的可执行二进制代码，能够满足对组件架构的所有需求。遵循COM的规范标准，组件与应用、组件与组件之间可以互操作，极其方便地建立可伸缩的应用系统。COM对象是对象的一个实例。

上一章描述的类的对象与本章所描述的“对象”不是同一个概念。类是抽象化的一类事物，类的对象是这类事物的一个具体体现；而“对象”是具体事物的集合。说白了类的对象是一个事物的实例，“对象”是多个实例事物的集合。

### 11.1.1 对象型变量的定义

易语言从3.7版开始，增加了一个新的数据类型“对象”（位于系统核心支持库），通过它就可以方便地引用对象数据类型。

（1）在易程序中，对象可以通过定义数据类型的实例来创建。定义方法主要有两种，将一个组件拖放到窗口上，或者在程序中定义一个具有该数据类型的变量。被拖放到设计窗口上的组件或者所定义的变量即为对象。一个数据类型可以定义出无限多个对象。

通过在对象及其属性或者命令之间加上句点（“.”）即可以引用对象的属性及命令。

例如：画板组件可以看作一个对象。

```
画板1.画笔颜色 = #红色  
画板1.画直线 (10, 10, 100, 100)
```

定义为“画板”数据类型的画板对象变量也是一个对象。将“画板1”组件赋值给画板对象变量，画板对象变量就拥有了“画板1”组件的所有属性、命令、事件。





变量名	类 型	静态	数组	备 注
画板对象	画板			

```
画板对象 = 画板1
画板对象.画笔颜色 = #红色
画板对象.画直线 (10, 10, 100, 100)
```

这两段代码完成的功能是一样的。

(2) 在易程序中，对象也可以通过一种抽象的定义方式来创建。该对象变量定义后并不知道具体的“对象”是什么，需要通过“对象”数据类型提供的命令来实现该“对象”的具体实例。定义一个类型为“对象”的对象变量，使用“创建”命令创建一个对象的实例。如下所示：

变量名	类 型	静态	数组	备 注
对象变量	对象			

```
对象变量.创建 (“对象类型”, )
```

### 11.1.2 “对象”数据类型

下面通过对Word应用程序的操作来学习数据类型“对象”的使用方法。在学习之前，先创建几个变量，以备介绍Word对象时用到。

变量名	类 型	静态	数组	备 注
Word对象	对象			
doc对象	对象			
range对象	对象			
变体型变量	变体型			

#### 1) “创建 ()” 命令

<逻辑型> 对象. 创建 (文本型 对象类型, [文本型 类型库文件名])

创建指定类型的 COM 对象，本对象中的原有内容将被释放。成功返回真，否则返回假。

参数<1>的名称为“对象类型”，类型为“文本型 (text)”。本参数指定对象的名称，或对应的GUID。

例如：“Word.Application”、“{000209FF-0000-0000-C000-000000000046}”、“Excel.Application”等均可。

参数<2>的名称为“类型库文件名”，类型为“文本型 (text)”，可以被省略。如果需要通过类型库才能访问指定对象，可在本参数中提供其类型库或类型库数据所在文件名。

**注意：**如果没有提供文件路径，且在当前目录下无法找到该文件，系统将自动进行搜索。如果省略本参数，默认为不需要类型库文件。





例如：创建Word对象，操作没有打开的Word。根据输出的“是否为空（）”命令的返回值判断Word对象是否创建成功。

Word对象.创建(“Word.Application”,)

输出调试文本(Word对象.是否为空())

## 2) “获取（）”命令

<无返回值> 对象.获取(文本型 对象类型)

获取当前操作系统中已经存在的指定类型 COM 对象，本对象中的原有内容将被释放。成功返回真，否则返回假。

参数<1>的名称为“对象类型”，类型为“文本型(text)”。本参数指定对象的类型文本。例如“Word.Application”、“{000209FF-0000-0000-C000-000000000046}”、“Excel.Application”等均可。

例如：获取Word对象，操作已经打开的Word。根据输出的“是否为空（）”命令的返回值判断Word对象是否创建成功。

Word对象.获取(“Word.Application”)

输出调试文本(Word对象.是否为空())

## 3) “清除（）”命令

<无返回值> 对象.清除()

将本对象的内容释放并清空。如果不调用本命令，则对象在退出其作用区域时会自动被释放。例如：假设对象存在于子程序局部变量中，当子程序调用退出时，该对象会被自动释放。

例如：先执行了“Quit”方法来关闭Word应用程序，再销毁Word对象。

Word对象.方法(“Quit”,)

Word对象.清除()

## 4) “是否为空（）”命令

<逻辑型> 对象.是否为空()

如果本对象的内容为空，返回真，否则返回假。

例如：根据输出的“是否为空（）”命令的返回值判断Word对象是否创建成功。

Word对象.创建(“Word.Application”,)

输出调试文本(Word对象.是否为空())

## 5) “是否相等（）”命令

<逻辑型> 对象.是否相等(对象 欲检查的对象)

如果本对象的内容与指定对象的内容相等，返回真，否则返回假。

参数<1>的名称为“欲检查的对象”，类型为“对象(ComObject)”。本参数指定欲检查其内容是否相等的对象。





例如：判断Word对象与打开的文档doc对象是否相等。

```
Word对象.创建 ("Word.Application", )
' Word对象.查看 ()
doc对象 = Word对象.读对象型属性 ("Documents", )
输出调试文本 (Word对象.是否相等 (doc对象))
```

#### 6) “读文本属性 ()” 命令

<文本型> 对象. 读文本属性 (文本型 属性名称, [通用型 参数数据], ...)

读取并返回本对象的指定文本类型属性值，如果失败将返回空文本，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型 (text)”。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：读取Word对象的名称，在输出面板中显示结果。

```
Word对象.创建 ("Word.Application", )
输出调试文本 (word对象.读文本属性 ("Name", ))
```

#### 7) “读数值属性 ()” 命令

<双精度小数型> 对象. 读数值属性 (文本型 属性名称, [通用型 参数数据], ...)

读取并返回本对象的指定数值类型属性值，如果失败将返回数值0，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型 (text)”。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：读取Word应用窗口的宽度，在输出面板中显示结果。

```
Word对象.创建 ("Word.Application", )
输出调试文本 (Word对象.读数值属性 ("Width", ))
```

#### 8) “读逻辑属性 ()” 命令

<逻辑型> 对象. 读逻辑属性 (文本型 属性名称, [通用型 参数数据], ...)

读取并返回本对象的指定逻辑型属性值，如果失败将返回假，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型 (text)”。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。





例如：读取Word应用窗口是否可视。在输出面板中显示结果。

Word对象.创建 (“Word.Application”,)

输出调试文本 (Word对象.读逻辑属性 (“Visible”,))

#### 9) “读日期属性 ()” 命令

<日期时间型> 对象. 读日期属性 (文本型 属性名称, [通用型 参数数据], ...)

执行本对象的指定命令并忽略其返回值，如果失败，紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型 (text)”。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：读取Word对象中文档的保存日期，在输出面板中显示结果。如果读取日期错误，使用“对象.取错误”命令，在输出面板中显示错误原因。

Word对象.创建 (“Word.Application”,)

输出调试文本 (Word对象.读日期属性 (“date”,))

输出调试文本 (Word对象.取错误())

#### 10) “读对象型属性 ()” 命令

<对象> 对象. 读对象型属性 (文本型 属性名称, [通用型 参数数据], ...)

读取并返回本对象的指定对象属性值，如果失败将返回内容为空的对象，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型 (text)”。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：判断Word对象与打开的文档doc对象是否相等。

Word对象.创建 (“Word.Application”,)

Word对象.查看 ()

doc对象 = Word对象.读对象型属性 (“Documents”,)

输出调试文本 (Word对象.是否相等 (doc对象))

#### 11) “读属性 ()” 命令

<变体型> 对象. 读属性 (文本型 属性名称, [通用型 参数数据], ...)

读取并返回本对象的指定属性值，本命令可以用作读取任意类型的属性。如果失败将返回类型值为空的变体型对象，且紧跟其后执行“取错误”命令将返回非空文本，否则将返回包含对应数据类型数据的变体型对象。





参数<1>的名称为“属性名称”，类型为“文本型（text）”。

参数<2>的名称为“参数数据”，类型为“通用型（all）”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：读取Word应用程序使用者的名称，放到变体型变量中，在输出面板中显示变体型变量的文本内容。

```
Word对象.创建 ("Word.Application",)
变体型变量 = Word对象.读属性 ("UserName",)
输出调试文本 (变体型变量.取文本 ())
```

## 12) “写属性 ()” 命令

<逻辑型> 对象. 写属性 (文本型 属性名称, [通用型 参数数据], ...)

设置本对象指定属性的值，如果失败，返回假，并且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“属性名称”，类型为“文本型（text）”。

参数<2>的名称为“参数数据”，类型为“通用型（all）”，可以被省略。本参数及本参数后的所有扩展参数提供读写属性所需要的所有数据。如果读写该属性不需要任何参数，应保持本参数为空。

例如：设置Word应用程序可视为真。在输出面板中显示“写属性”的结果。

```
Word对象.创建 ("Word.Application",)
输出调试文本 (Word对象.写属性 ("Visible",真))
```

## 13) “方法 ()” 命令

<无返回值> 对象. 方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象的指定方法并忽略其返回值，如果失败，紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型（text）”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型（all）”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数，应保持本参数为空。

例如：退出Word应用程序，使用“Word对象.清除”命令销毁Word对象。

```
Word对象.方法 ("Quit",)
Word对象.清除 ()
```

## 14) “通用方法 ()” 命令

<变体型> 对象. 通用方法 (文本型 方法名称, [通用型 参数数据], ...)





执行本对象的指定方法并返回一个变体型对象值，该对象内记录该方法的返回数据。本命令可以用作执行返回任何数据类型数据的方法，如果该方法没有返回数据，所返回变体型对象的类型值将为空。如果失败，将返回一个类型值为空的变体型对象，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型(text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型(all)”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数，应保持本参数为空。

例如：取得一个打开的Word应用程序的Word对象，执行Word对象的通用方法，运行“宏”。

Word对象.获取 (“Word.Application”)

Word对象.通用方法 (“Run”, “宏名称”)

#### 15) “文本方法 ()” 命令

<文本型> 对象. 文本方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象返回文本类型数据的方法，如果失败将返回空文本，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型(text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型(all)”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数，应保持本参数为空。

例如：取得一个打开的Word应用程序的Word对象，执行Word对象的文本方法，取word文档主题格式名称。

Word对象.获取 (“Word.Application”)

Word对象.文本方法 (“GetDefaultTheme”, “wdDocument”)

#### 16) “数值方法 ()” 命令

<双精度小数型> 对象. 数值方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象返回数值类型数据的方法，如果失败将返回数值0，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型(text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型(all)”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参





数，应保持本参数为空。

例如：取得一个打开的Word应用程序的Word对象，执行Word对象的数值方法，设置键盘的输入法。

Word对象.获取 (“Word.Application”)

Word对象.数值方法 (“Keyboard”, 0)

#### 17) “逻辑方法 ()” 命令

<逻辑型> 对象. 逻辑方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象返回逻辑型数据的方法，如果失败将返回假，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型 (text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数，应保持本参数为空。

例如：取得一个打开的Word应用程序的Word对象，执行Word对象的逻辑方法，检查参数文本的拼写。

Word对象.获取 (“Word.Application”)

Word对象.逻辑方法 (“CheckSpelling”, “Word”)

#### 18) “日期方法 ()” 命令

<日期时间型> 对象. 日期方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象返回日期时间型数据的方法，如果失败将返回100年1月1日，且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”，类型为“文本型 (text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”，类型为“通用型 (all)”，可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数，应保持本参数为空。

例如：创建Word对象，设置Word应用程序可视，读取Word对象的文档doc对象，添加文档，执行doc对象的对象型方法，取段落range对象，向文档中当前段落插入日期。

Word对象.创建 (“Word.Application”, )

Word对象.写属性 (“Visible”, 真)

doc对象 = Word对象.读对象型属性 (“Documents”, )

doc对象 = doc对象.读对象型属性 (“add”, )

range对象 = doc对象.对象型方法 (“Range”, )

range对象.日期方法 (“InsertDateTime”, , , 真, , )





“InsertDateTime”具体使用方法可参见随书光盘例程。

### 19) “对象型方法 ()”命令

<对象> 对象. 对象型方法 (文本型 方法名称, [通用型 参数数据], ...)

执行本对象返回对象型数据的方法, 如果失败将返回内容为空的对象, 且紧跟其后执行“取错误”命令将返回非空文本。

参数<1>的名称为“方法名称”, 类型为“文本型(text)”。本参数提供欲调用方法的名称。

参数<2>的名称为“参数数据”, 类型为“通用型(all)”, 可以被省略。本参数及本参数后的所有扩展参数提供调用方法所需要的所有数据。如果该方法不需要任何参数, 应保持本参数为空。

例如: 创建Word对象, 设置Word应用程序可视, 读取Word对象的文档doc对象, 添加文档, 执行doc对象的对象型方法, 取段落range对象, 向文档中当前段落写入文本。

```
Word对象.创建 ("Word.Application", )
Word对象.写属性 ("Visible", 真)
doc对象 = Word对象.读对象型属性 ("Documents", )
doc对象 = doc对象.读对象型属性 ("add", )
range对象 = doc对象.对象型方法 ("Range", )
range对象.写属性 ("Text", "易语言")
```

### 20) “创建图片对象 ()”命令

<逻辑型> 对象. 创建图片对象 (字节集 图片数据)

为指定图片数据创建对应的 COM 图片对象, 本对象中的原有内容将被释放。

参数<1>的名称为“图片数据”, 类型为“字节集(bin)”。本参数提供欲用作创建图片对象的图片数据。

例如: 使用图片对象变量载入一个图片并创建该图片对象。

图片对象变量.创建图片对象(读入文件("C:\图片.bmp"))

### 21) “创建字体对象 ()”命令

<逻辑型> 对象. 创建字体对象 (字体 字体数据)

为指定字体数据创建对应的 COM 字体对象, 本对象中的原有内容将被释放。

参数<1>的名称为“字体数据”, 类型为“字体(font)”。本参数提供欲用作创建字体对象的字体数据。

例如: 使用字体对象变量载入字体数据并创建该字体对象。

字体对象变量.创建字体对象(字体数据)

### 22) “取回图片 ()”命令

<字节集> 对象. 取回图片 ()

如果本对象为 COM 图片对象, 则取回其具体图片数据。成功返回图片数据字节集,





失败返回空字节集。

例如：取回图片数据，放到图片数据变量中。

图片数据变量 = 图片对象变量.取回图片 ()

23) “取回字体 ()” 命令

<字体> 对象. 取回字体 ()

如果本对象为 COM 字体对象，则取回其具体字体数据。如果失败，紧跟其后执行“取错误”命令将返回非空文本。

例如：取回字体数据，放到字体数据变量中。

字体数据变量 = 字体对象变量.取回字体 ()

24) “取错误 ()” 命令

<文本型> 对象. 取错误 ()

当读写对象属性、执行对象方法或取回字体时，紧跟该语句后执行本命令可以检查其是否执行成功。如果成功，本命令将返回空文本，如果失败，本命令将返回一个描述具体错误信息的非空文本。

例如：取Word对象错误信息，在输出面板中显示结果。

输出调试文本(Word对象.取错误 ())

25) “查看 ()” 命令

<无返回值> 对象. 查看 ()

本命令仅在易程序的调试版本中被执行，在发布版本中将被直接跳过。通过对话框的方式查看本对象的调用格式信息，便于编写相关程序，如图11-1所示。

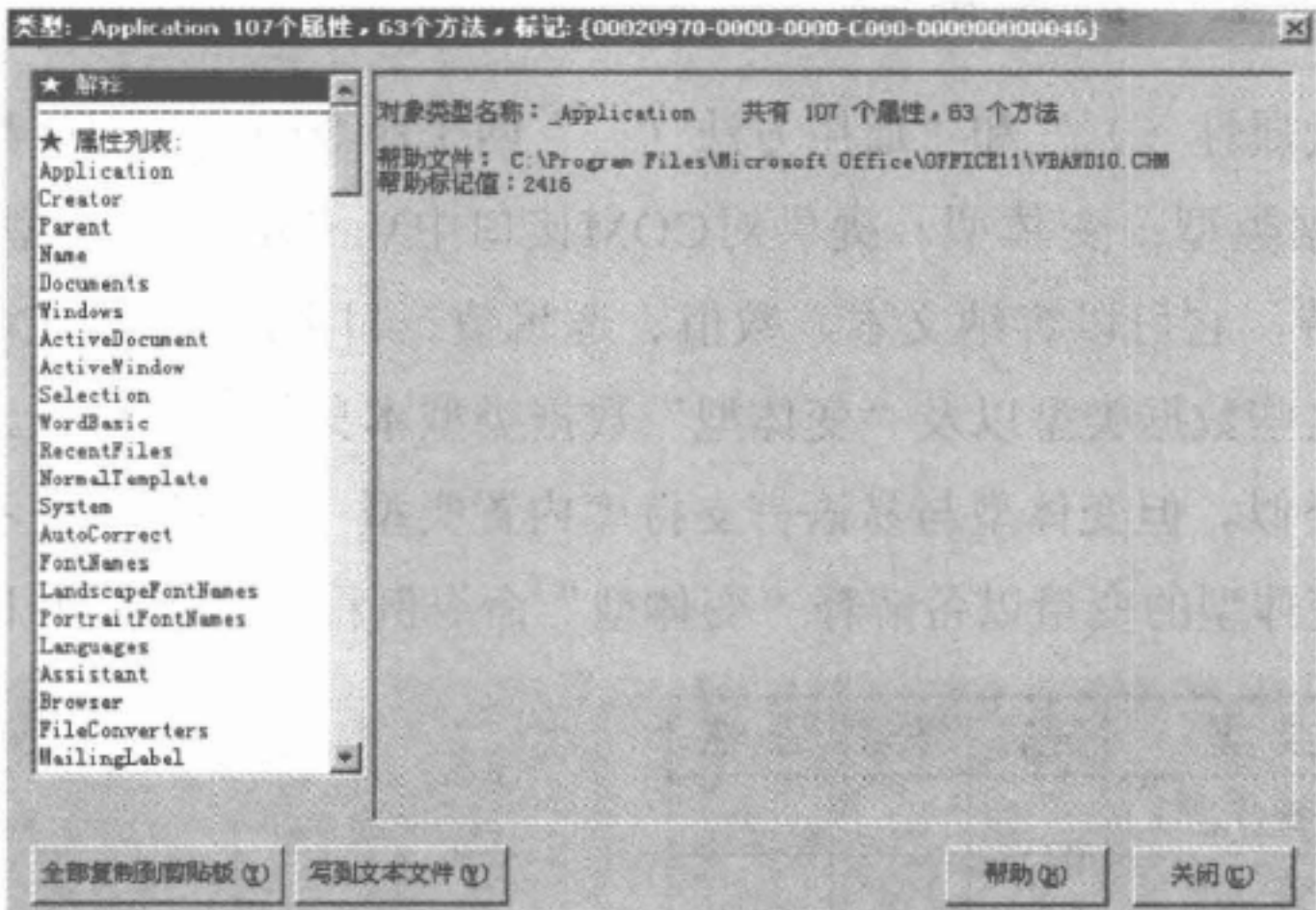


图11-1 “对象. 查看 ()” 执行效果图

可以看到在此对话框中，列出了该对象的所有基本信息，包括属性列表、方法列表，以及各属性、方法在易语言中的调用方式。有了这些信息，即使没有其他参考资料，也能大致了解该对象的使用方法。





## 26) “取接口 ()” 命令

<逻辑型> 对象. 取接口 (通用型 对象或窗口组件, [文本型 接口标志符])

获取指定对象或者OCX窗口组件中的指定接口，本对象中的原有内容将被释放。成功返回真，失败返回假。

参数<1>的名称为“对象或窗口组件”，类型为“通用型 (all)”。本参数提供欲获取其接口对象的对象数据，其数据类型为以下之一：1 (COM包装库封装出来的对象型数据类型)；2 [核心支持库中的“对象”数据类型 (即本数据类型)]；3 (COM包装库封装出来的OCX窗口组件数据类型)。

参数<2>的名称为“接口标志符”，类型为“文本型 (text)”，可以被省略。可以在本参数中指定欲获取接口的具体名称或其全球唯一标志符 (GUID)，如“IDataSourceControl”、{B62EE548-2B65-43BE-9F39-E2B71742578A}等。如果本参数被省略，则默认取出对象的基本接口。

例如：取Word对象的接口，放到对象变量中。

变量名	类型	静态	数组	备注
对象变量	对象			

Word对象. 创建 (“Word.Application”, )

对象变量. 取接口 (Word对象, )

对象变量. 查看 ()

Word对象及其所属的其他对象的属性、方法的具体使用，可以在网络上搜索查找。

## 11.1.3 “变体型”数据类型

在对象的“读属性 ()”和“通用方法 ()”两个命令中，返回值都是“变体型”。这又引出一种数据类型：变体型。提供对COM接口中Variant数据类型 (即VB中的Any数据类型) 的支持，它可以容纳文本、数值、逻辑值、日期时间值、COM对象型数据类型，还可以容纳这些数据类型以及“变体型”数据类型本身的数组形式。其使用的方法和对象的使用方法类似。但变体型与易语言支持库内置类型“通用型”是不同的。在学习之前，先创建关于变异型的变量以备解释“变体型”命令例程时用到。如下所示：

变量名	类型	静态	数组	备注
变体型变量	变体型			
变体型数组变量	变体型		0	
变体类型变量	变体类型			

## 1) “清除 ()” 命令

<无返回值> 对象. 清除 ()

将本对象的内容释放并清空。如果不调用本命令，则对象在退出其作用区域时会自动



被释放。

例如：释放变体型变量。

变体型变量.清除 ()

2) “取类型 ()” 命令

<变体类型> 对象.取类型 ()

取回当前变体型对象的数据类型。

例如：取变体型变量内容的数据类型，放到变体类型变量中，在输出面板中显示结果。

变体类型变量 = 变体型变量.取类型 ()

输出调试文本(变体类型变量)

数据类型名称：变体类型

本数据类型为枚举常量集合类型，共包含 15 个枚举常量值。见表11-1。

表11-1 变体类型枚举常量

枚举常量值名称	枚举常量值	枚举常量值名称	枚举常量值
未知	-1	数值型数组	7
空	0	文本型数组	8
数值型	1	逻辑型数组	9
文本型	2	日期型数组	10
逻辑型	3	对象型数组	11
日期型	4	错误值型数组	12
对象型	5	变体型数组	13
错误值型	6		

3) “取数组成员数 ()” 命令

<整数型> 对象.取数组成员数 ()

如果本对象中存放的是数组数据，执行本命令将返回其成员数目，如果不是数组数据，执行本命令将返回 -1。

例如：取数组型的变体型数组变量的成员数，在输出面板中显示结果。

输出调试文本 (变体型数组变量.取数组成员数 ())

4) “取文本 ()” 命令

<文本型> 对象.取文本 ([整数型 成员索引])

返回本对象中或本对象数组成员中的文本数据，如果当前数据的数据类型不为文本型，将自动进行转换，如果转换失败将返回空文本。

参数<1>的名称为“成员索引”，类型为“整数型 (int)”，可以被省略。如果当前对象为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从 1 开始。除上述情况外，省略本参数。





例如：取变体型变量的文本内容。在输出面板中显示结果。

输出调试文本（变体型变量.取文本（））

#### 5) “取数值（）”命令

<双精度小数型> 对象.取数值（[整数型 成员索引]）

返回本对象中或本对象数组成员中的数值数据，如果当前数据的数据类型不为数值或错误值型，将自动进行转换，如果转换失败将返回0。

参数<1>的名称为“成员索引”，类型为“整数型（int）”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的数值内容。在输出面板中显示结果。

输出调试文本（变体型变量.取数值（））

#### 6) “取逻辑值（）”命令

<逻辑型> 对象.取逻辑值（[整数型 成员索引]）

返回本对象中或本对象数组成员中的逻辑值数据，如果当前数据的数据类型不为逻辑型，将自动进行转换，如果转换失败将返回假。

参数<1>的名称为“成员索引”，类型为“整数型（int）”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的逻辑值内容。在输出面板中显示结果。

输出调试文本（变体型变量.取逻辑值（））

#### 7) “取日期（）”命令

<日期时间型> 对象.取日期（[整数型 成员索引]）

返回本对象中或本对象数组成员中的日期时间值数据，如果当前数据的数据类型不为日期时间型，将自动进行转换，如果转换失败将返回100年1月1日。

参数<1>的名称为“成员索引”，类型为“整数型（int）”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的日期内容。在输出面板中显示结果。

输出调试文本（变体型变量.取日期（））

#### 8) “取对象（）”命令

<对象> 对象.取对象（[整数型 成员索引]）

返回本对象中或本对象数组成员中的COM对象型数据，如果当前数据的数据类型不为COM对象型，将返回空COM对象。

参数<1>的名称为“成员索引”，类型为“整数型（int）”，可以被省略。如果当



前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的对象内容，放到对象变量中。

对象变量 = 变体型变量.取对象 ( )

#### 9) “取变体型 ( )” 命令

<变体型> 对象. 取变体型 ( [整数型 成员索引] )

返回本对象中或本对象数组成员中的变体型数据类型数据，如果当前数据的数据类型不为变体型，将自动进行转换。

参数<1>的名称为“成员索引”，类型为“整数型 (int)”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的变体型内容。放到变体型变量1中。

变体型变量1 = 变体型变量.取变体型 ( )

#### 10) “赋值 ( )” 命令

<逻辑型> 对象. 赋值 ( [通用型数组/非数组 欲写入数据] , [整数型 成员索引] )

设置本对象的内容，成功返回真，失败返回假。

参数<1>的名称为“欲写入数据”，类型为“通用型 (all)”，可以被省略，提供参数数据时可以同时提供数组或非数组数据。参数值可以为文本、数值、逻辑值、日期时间值、COM对象型数据类型或者这些数据类型及“变体型”数据类型本身的数组形式。如果本参数被省略，则将对象清空。

参数<2>的名称为“成员索引”，类型为“整数型 (int)”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：设置变体型变量的内容。

变体型变量.赋值 (“易语言”, )

#### 11) “创建数组 ( )” 命令

<逻辑型> 对象. 创建数组 ( 整数型 成员类型, 整数型 成员数目 )

创建指定类型和指定成员数的空白数组，对象中的原有内容将被清空，成功返回真，失败返回假。

参数<1>的名称为“成员类型”，类型为“整数型 (int)”。指定所创建数组成员的数据类型，为以下数值之一：1 (字节型)；2 (短整数型)；3 (整数型)；4 (小数型)；5 (双精度小数型)；6 (文本型)；7 (逻辑型)；8 (日期型)；9 (对象型)；10 (变体型)。

参数<2>的名称为“成员数目”，类型为“整数型 (int)”。指定所创建数组成员的数目。





例如：创建变体型变量的包含2个成员的空白整数型数组。

变体型变量.创建数组(3, 2)

12) “置类型 ()” 命令

<逻辑型> 对象. 置类型 (变体类型 欲设置的类型)

清除当前变体型对象中的原有内容，并改变它的数据类型，成功返回真，失败返回假。

参数<1>的名称为“欲设置的类型”，类型为“变体类型 (VariantType)”。参数值提供欲设置的变体类型。

例如：设置变体型变量内容的数据类型为文本型。

变体型变量.置类型(变体类型变量.文本型)

13) “取字节集 ()” 命令

<字节集> 对象. 取字节集 ([整数型 成员索引])

返回本对象中或本对象数组成员中的字节集数据，如果当前数据的数据类型不为字节集型，将自动进行转换，如果转换失败将返回空字节集。

参数<1>的名称为“成员索引”，类型为“整数型 (int)”，可以被省略。如果当前对象内为数组型数据，且欲对数组内的某个成员进行操作，则可以在此参数中指定其引用索引值，该索引值从1开始。除上述情况外，省略本参数。

例如：取变体型变量的字节集内容，放到字节集变量中。

字节集变量 = 变体型变量.取字节集 ()

## 11.2 COM对象的应用

COM是微软公司为了计算机工业的软件生产更加符合人类的行为方式开发的一种新的软件开发技术。在COM构架下，人们可以开发出各种各样的功能专一的组件，然后将它们按照需要组合起来，构成复杂的应用系统。由此带来的好处是多方面的：可以将系统中的组件用新的替换掉，以便随时进行系统的升级和定制；可以在多个应用系统中重复利用同一个组件；可以方便地将应用系统扩展到网络环境下；COM与语言平台无关的特性使所有的程序员均可充分发挥自己的才智与专长编写组件模块等。

COM (Component Object Model) 组件对象模型实际上是一些小的二进制可执行程序，它们可以给应用程序、操作系统以及其他组件提供服务。

在COM规范中，没有COM对象的严格定义，COM组件提供给客户的是以对象形式封装起来的实体，客户与组件交互的实体是COM对象。COM对象有自己的属性和方法，但这些都COM封装了起来，客户只有通过接口才能对COM的方法进行调用，COM接口是一个包





含一个函数指针数组的内存结构。每一个数组元素包含的是一个由组件所实现的函数地址。组件与组件之间、组件与客户之间都要通过接口进行交互。接口是COM与外界通信、交互的唯一途径。

在Windows系统下，有大量的COM对象存在，尤其是Windows本身就是一个庞大的COM对象库。充足的组件给程序开发人员提供了极大便利。

下面，针对COM组件中的Excel对象来做一个实例：打开一个Excel文件。

第一步：定义两个“对象”型的变量

窗口程序集名	保 留	保 留	备 注
窗口程序集1			
变量名	类 型	数 组	备 注
excel对象	对象		
工作表对象	对象		

第二步：创建Excel对象

```
excel对象.创建 (“excel.application”,)
```

第三步：查看Excel对象的属性和方法

```
excel对象.查看 ()
```

如果对要操作的对象已非常熟悉，或有足够的参考资料，可以省略这一步。

第四步：读写对象属性，调用对象方法

```
excel对象.写属性 (“Visible”, 真)
```

工作表对象 = excel对象.读对象型属性 (“workbooks”,)

工作表对象.方法 (“open”, 取运行目录 () + “\Book1.xls”)

第五步：清除对象，先退出Excel程序再清除Excel对象

```
excel对象.方法 (“quit”,)
```

```
excel对象.清除 ()
```

在程序编写过程中，用到了属性和方法的名称，都是使用“查看”命令查到的。

完整的代码如下：

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

```
excel对象.创建 (“excel.application”,)
excel对象.写属性 (“visible”, 真)
工作表对象 = excel对象.读对象型属性 (“workbooks”,)
工作表对象.方法 (“open”, 取运行目录 () + “\Book1.xls”)
```

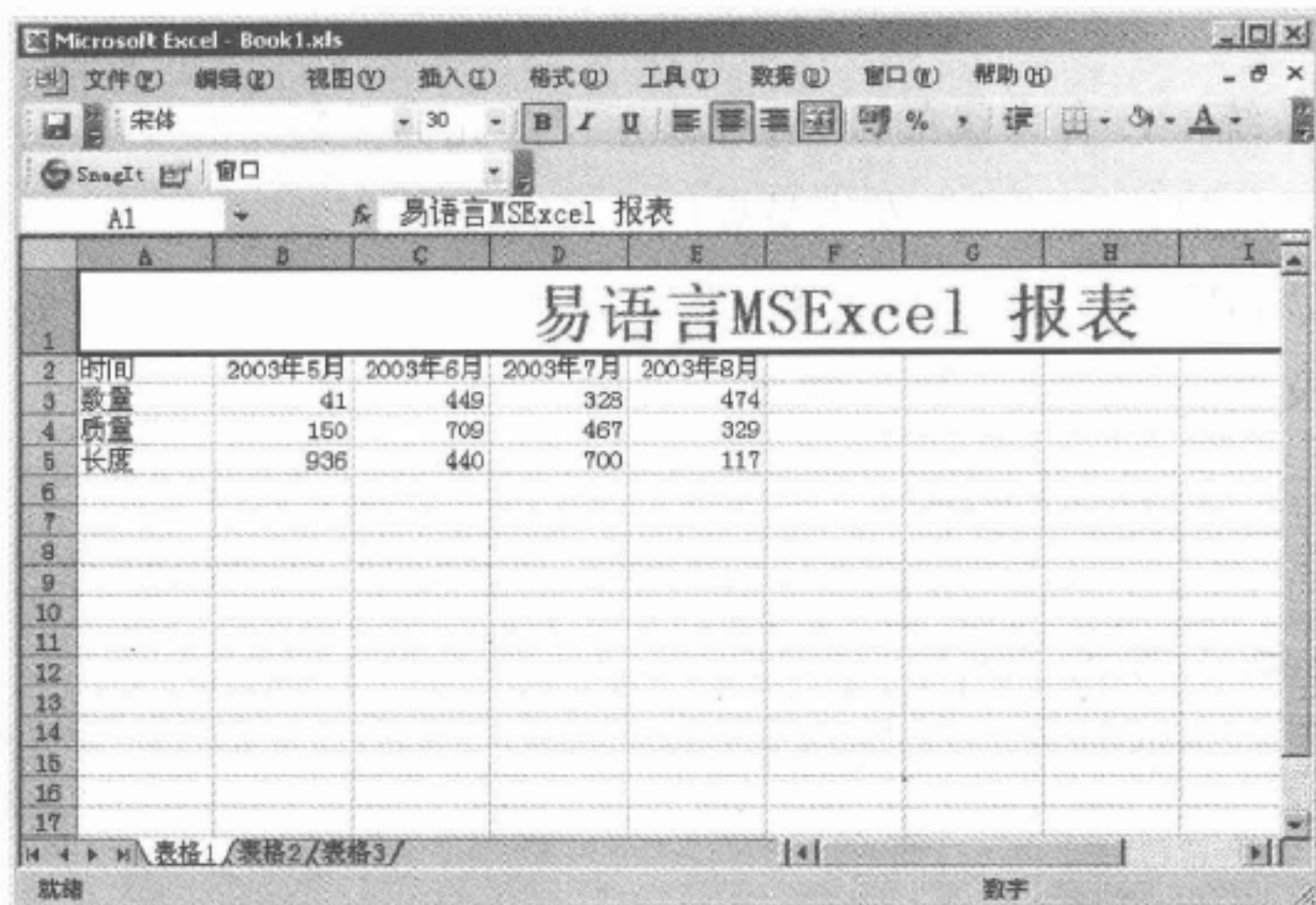
子程序名	返回值类型	公开	备 注
_启动窗口_将被销毁			

```
excel对象.方法 (“Quit”,)
excel对象.清除 ()
```





程序已经编写完毕，按F5键运行程序，单击窗口中的“打开Excel”按钮，会发现Excel已自动启动，并打开了指定的表格，如图11-2所示。



易语言MSEExcel 报表				
时间	2003年5月	2003年6月	2003年7月	2003年8月
数量	41	449	328	474
质量	150	709	467	329
长度	936	440	700	117

图11-2 “COM对象”打开Excel执行效果图

Excel对象及其所属的其他对象的属性、方法的使用方法，可以在网络上搜索查找。

一个对象的使用必须先知道该对象的对象类型名称，然后调用“查看”命令查看该对象的属性、方法及所属子对象的使用方法，逐步掌握该对象的全部用法。

## 11.3 小结

任何事物都可以是一个对象。对象具有自身的属性、使用的命令和产生的事件。COM对象是具有规范接口的命令的集合。COM对象是对象的一个实例，通过“对象”可以使用“COM对象”，通过“对象.查看”可以了解对象具体的属性、方法、事件。

## 11.4 习题

- 11-1 对象是\_\_\_\_\_。
- 11-2 COM组件对象模型是\_\_\_\_\_。
- 11-3 COM对象是对象的\_\_\_\_\_。
- 11-4 COM与外界通信、交互的唯一途径是\_\_\_\_\_。
- 11-5 使用COM对象对Word组件接口进行调用，掌握Word的基本功能。
- 11-6 使用“对象.查看（）”命令，查看Excel组件的属性和方法。







## 第十二章 易模块的应用

### 本章目标

在本章结束时，我们能够：

- 了解什么是模块
- 了解怎样创建易模块
- 了解怎样引用易模块
- 掌握使用易模块编程的方法





## 12.1 了解易模块

模块是为完成某一功能所需的一段程序。一般是将一些通用的功能集成在一起，供其他程序调用。可简化软件系统的复杂度，具有独立功能，可复制，可拆分，能够重复利用。

易语言也提供了模块化的开发支持——“易模块”。易模块是一段特殊格式的易代码，所有代码都封装在模块中，模块的使用者只能看到公开的接口结构，无法看到实现功能的代码。通过使用易模块，编程人员可以将常用的易代码封装起来并重复使用到其他易程序中，或提供给第三方用于易语言程序的开发。

## 12.2 易模块的开发与编译

### 12.2.1 易模块的开发

可以根据需要来创建易模块，并在易语言中引用。下面就通过一个简单的例程来了解一下易模块的创建方法。

第一步：在易语言“新建”窗口中选择“Windows易语言模块”，然后点击“确定”按钮，如图12-1所示。

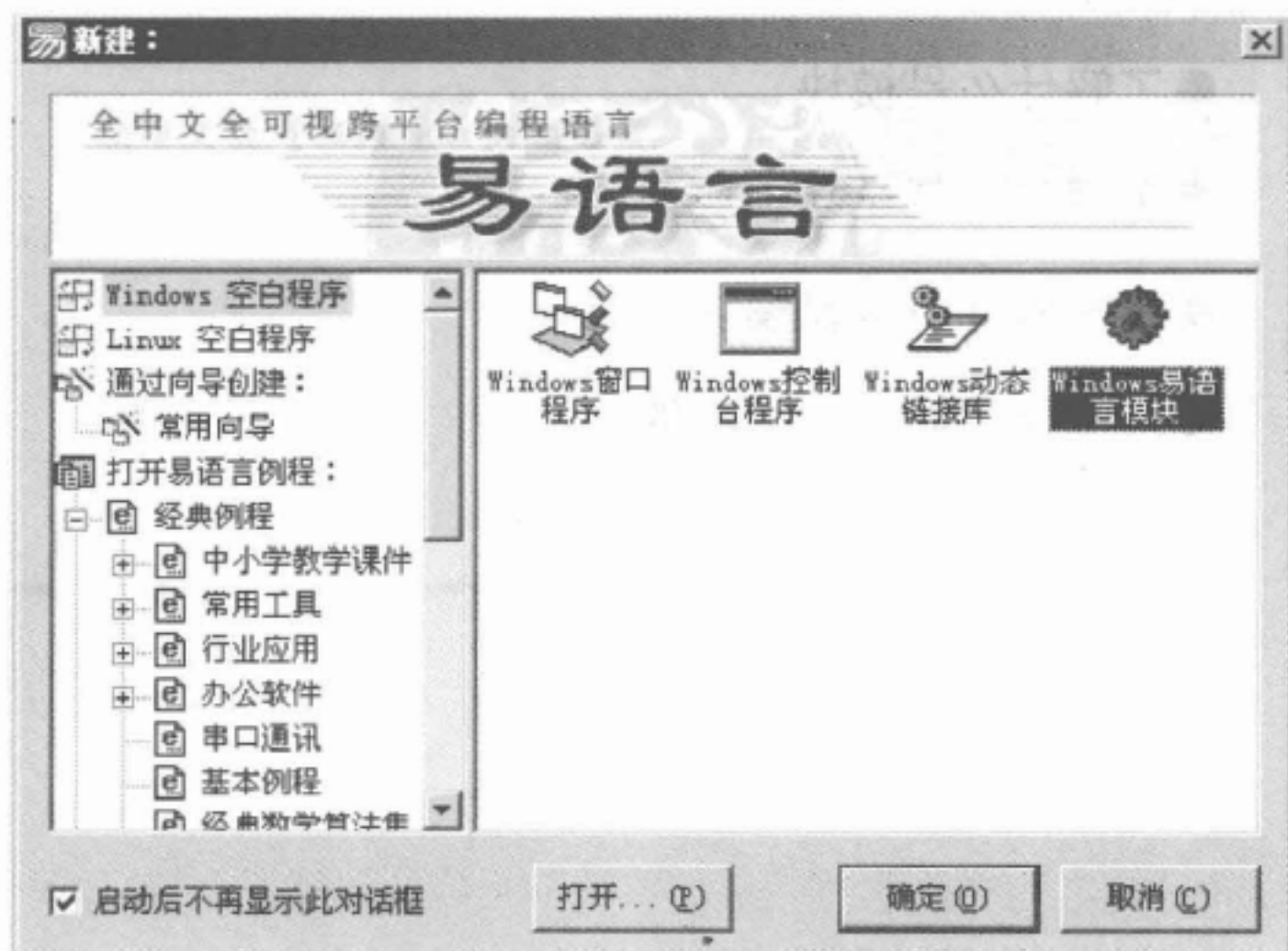


图12-1 新建Windows易语言模块

进入代码编辑区，易语言会自动创建两个子程序。代码如下：





子程序名	返回值类型	公开	备注
_启动子程序	整数型		请在本子程序中放置易模块初始化代码

\_临时子程序 () ' 在初始化代码执行完毕后调用测试代码  
返回 (0) ' 可以根据您的需要返回任意数值

子程序名	返回值类型	公开	备注
_临时子程序			

' 本名称子程序用作测试程序用，仅在开发及调试环境中有效，编译发布程序前将被系统自动清空，请将所有用作测试的临时代码放在本子程序中。 \*\*\*注意不要修改本子程序的名称、参数及返回值类型。

“\_启动子程序”：此子程序负责进行易模块的初始化。

“\_临时子程序”：用做调试易模块。所有在该子程序中书写的程序代码，仅在调试环境下运行有效。在编译易模块时，该子程序中的所有代码将不会被编译到模块文件中。

第二步：在此程序集中新建一个子程序，改名为“求阶乘”，并将“公开”选中，在易模块中任何程序集中选中“公开”的子程序都将作为对外的接口。然后输入以下代码：

子程序名	返回值类型	公开	备 注		
求阶乘	整数型	✓			
参数名	类 型	参考	可空	数组	备 注
阶乘数	整数型				

变量名	类型	静态	数组	备注
阶乘结果	整数型			
计次变量	整数型			

```
--- 如果真 (阶乘数 = 0)
  返回 (1) ' 0的阶乘等于1
阶乘结果 = 1
---> 计次循环首 (阶乘数, 计次变量)
  阶乘结果 = 阶乘结果 × 计次变量
--- 计次循环尾 ()
返回 (阶乘结果)
```

代码完成后，可以在易模块的状态下对代码的功能进行调试。

在“\_临时子程序”中调用易模块子程序“求阶乘”。代码如下：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		请在本子程序中放置易模块初始化代码

\_临时子程序 () ' 在初始化代码执行完毕后调用测试代码  
返回 (0) ' 可以根据您的需要返回任意数值

子程序名	返回值类型	公开	备注
_临时子程序			

' 本名称子程序用作测试程序用，仅在开发及调试环境中有效，编译发布程序前将被系统自动清空，请将所有用作测试的临时代码放在本子程序中。 \*\*\*注意不要修改本子程序的名称、参数及返回值类型。

输出调试文本 (求阶乘 (5))

第三步：按F5运行程序，可以在输出面板中看到整数5的阶乘的结果。

以上就是一个简单的“易模块”的创建过程。





## 12.2.2 易模块的编译

易模块创建完成后，要通过编译才可供其他易程序调用。

第一步：在“程序”菜单中选择“配置”选项，如图12-2所示。

第二步：在弹出的“程序配置”对话框中添写相应程序配置的内容，如图12-3所示。

对话框中各选项卡的含义如下：

- 通常：程序的名称、描述、备注、版本号等。
- 作者信息：作者的名称、地址、联系方式、版权声明等。
- 其它：修改或设置源代码的密码。

第三步：选择菜单“编译”→“编译”选项，对所做的易模块进行编译。在弹出的保存对话框中，保存文件名为“求阶乘”，易模块文件的后缀是“.ec”，如图12-4所示。

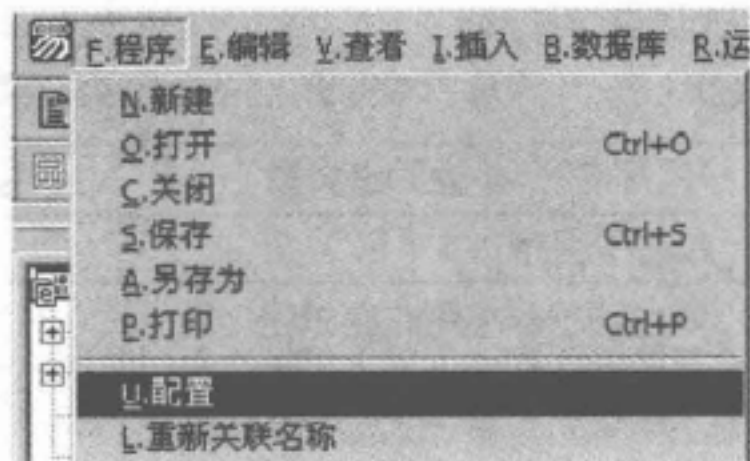


图12-2 选择菜单“程序”→“配置”

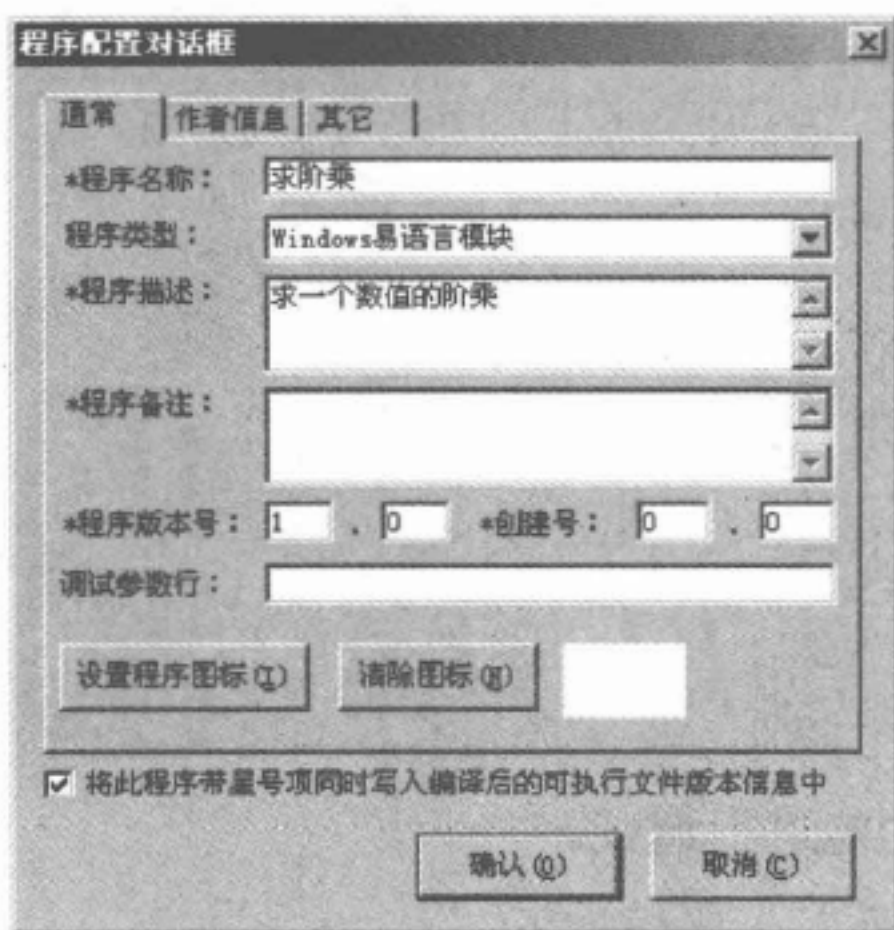


图12-3 模块程序配置

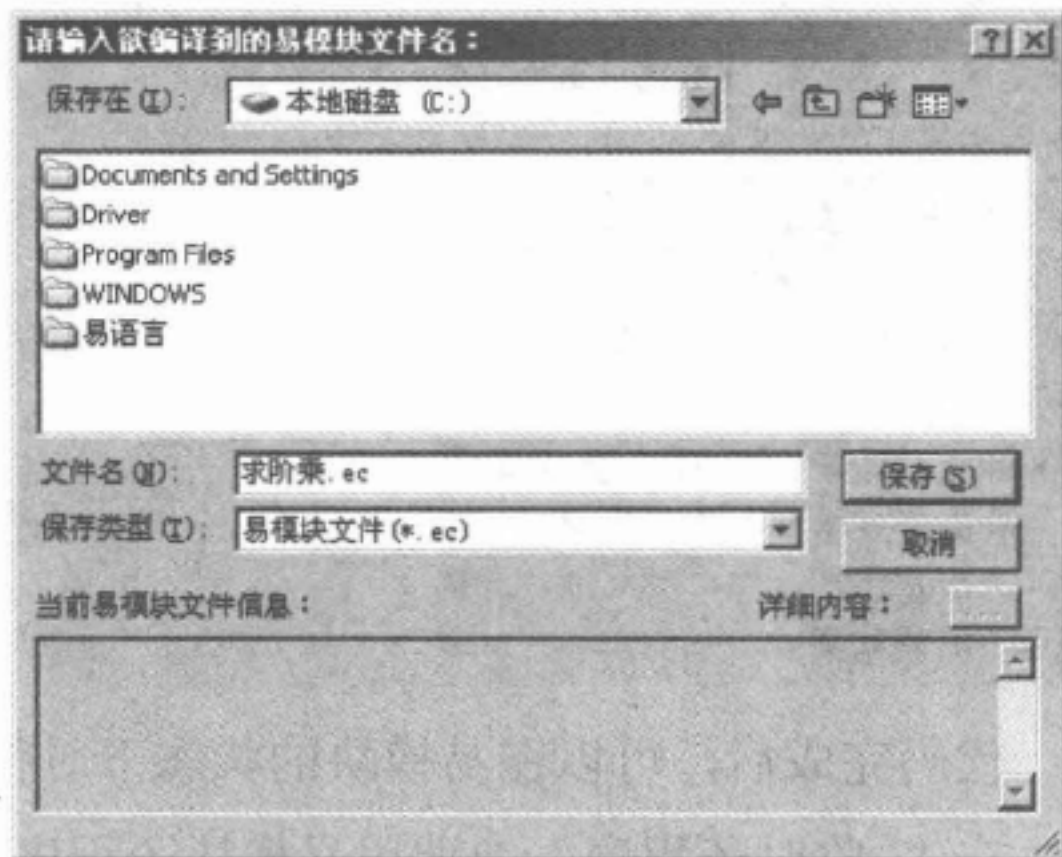


图12-4 保存模块

在弹出的“询问”信息框中，提示易模块编译成功，并询问是否将当前编译好的易模块程序导入到易语言程序系统的模块库中，选择“是”。（也可以通过“模块引用表”进行导入。参见“12.3 易模块的引用方法”。）如图12-5所示。

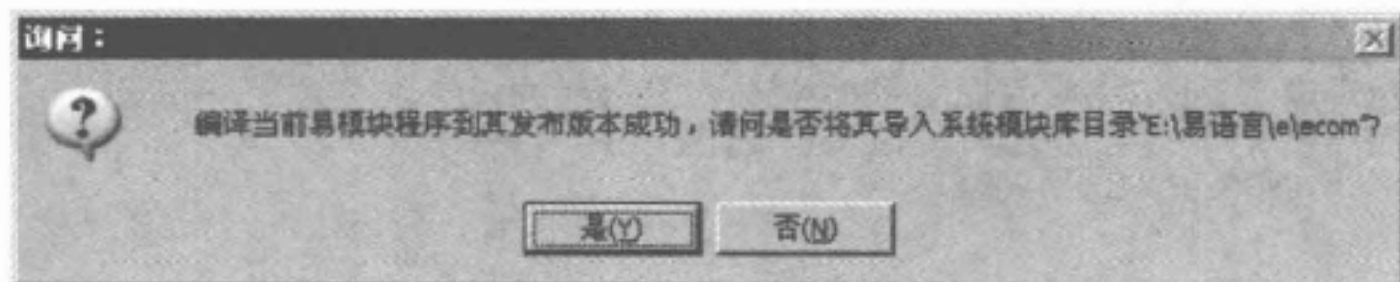


图12-5 “询问”信息框

至此，一个简单的易模块的编写和编译就完成了。





### 12.3 易模块的引用方法

对于已经编译好的易模块，要正确引用到易语言系统中才可以被调用。下面来了解易模块的引用方法。

第一步：新建一个易程序，在“程序”面板中选择“模块引用表”，如图12-6所示。

右键单击“模块引用表”，选择“添加模块引用”或双击“模块引用表”，如图12-7所示。

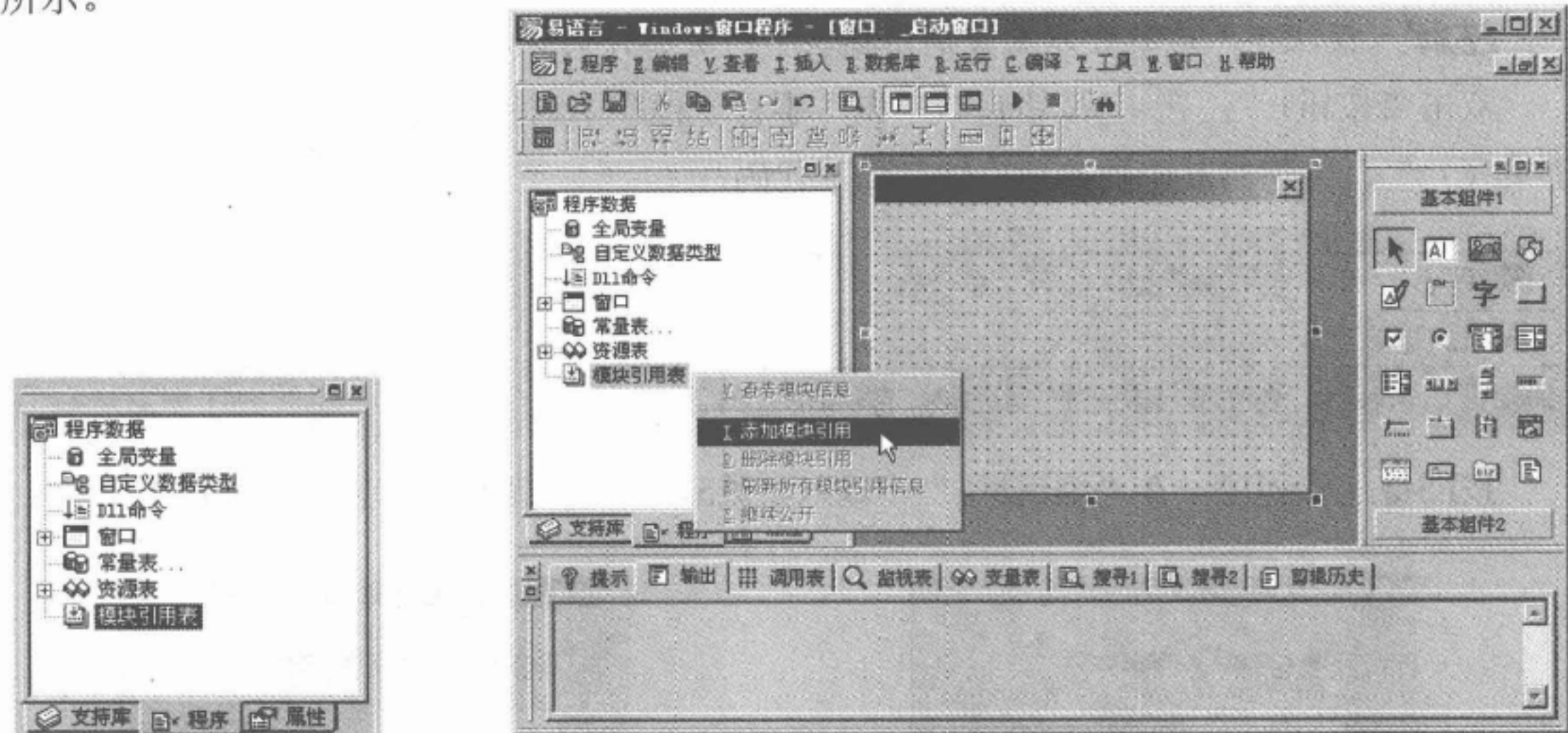


图12-6 选择模块引用表

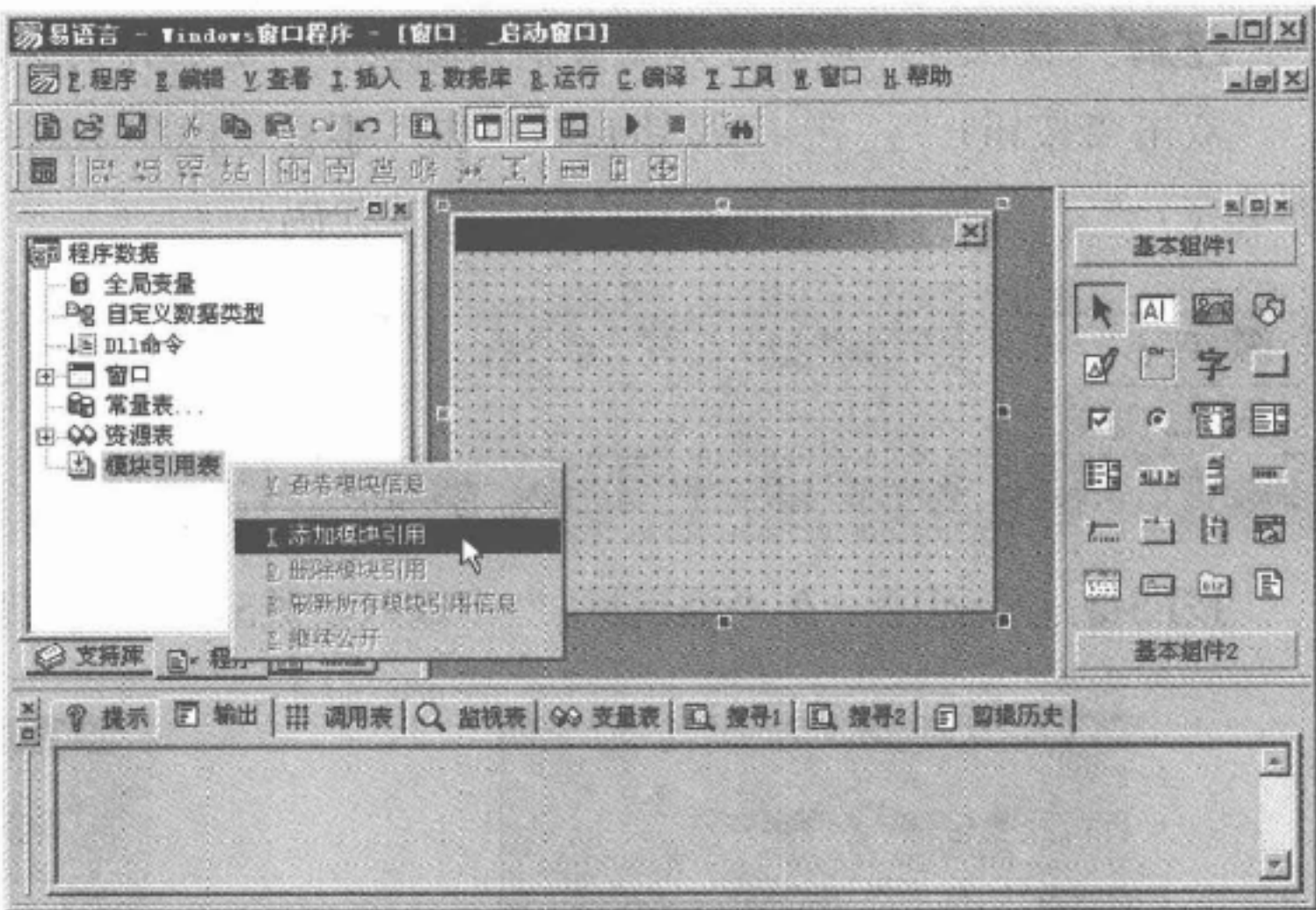


图12-7 添加模块引用

在弹出的易模块选择对话框中，选择“求阶乘.ec”文件（易模块文件），此文件可在随书光盘的本章目录中找到，如图12-8所示。

第二步：选择打开易模块后，在“模块引用表”中将出现该模块，如图12-9所示。

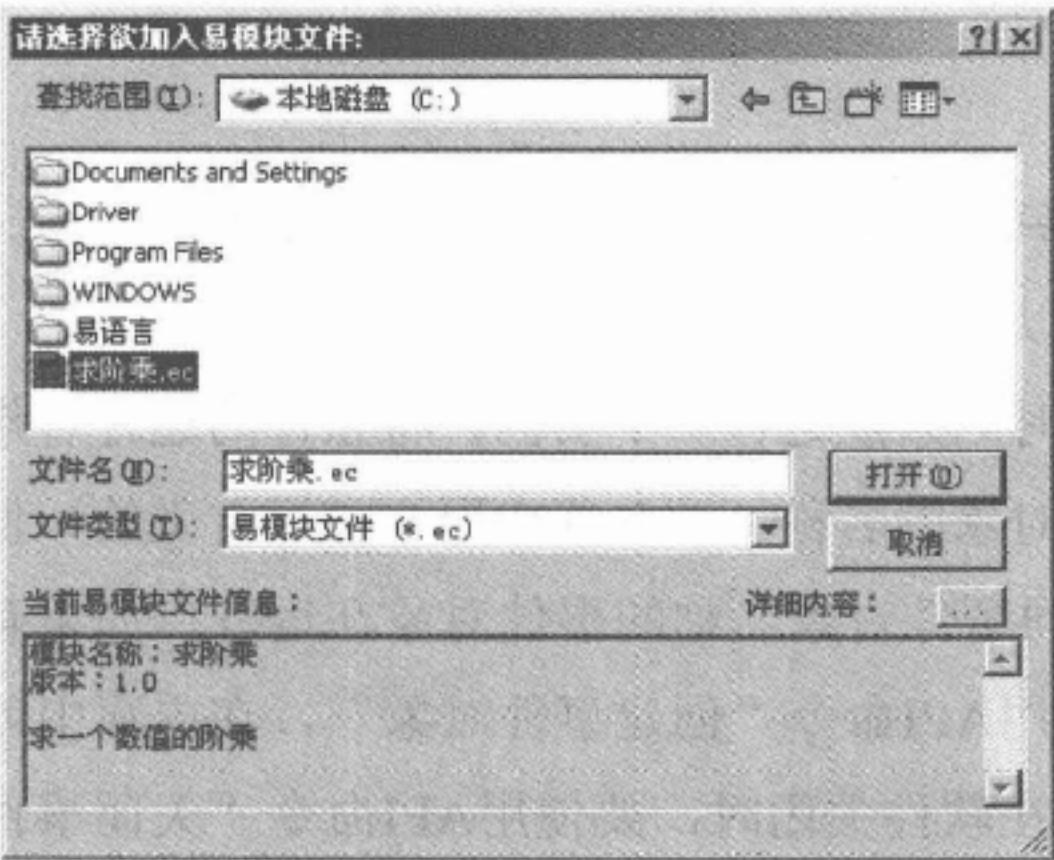


图12-8 易模块选择对话框



图12-9 导入易模块之后的“模块引用表”





双击模块名，弹出“易模块公开信息”对话框。左边的模块列表列出了模块中所有公开的可调用的子程序名称；右边的“信息”框中显示的是所选模块的说明和模块中接口的调用说明。对于没有说明文档的模块，可以通过此公开信息了解该模块接口的最基础信息，如图12-10所示。

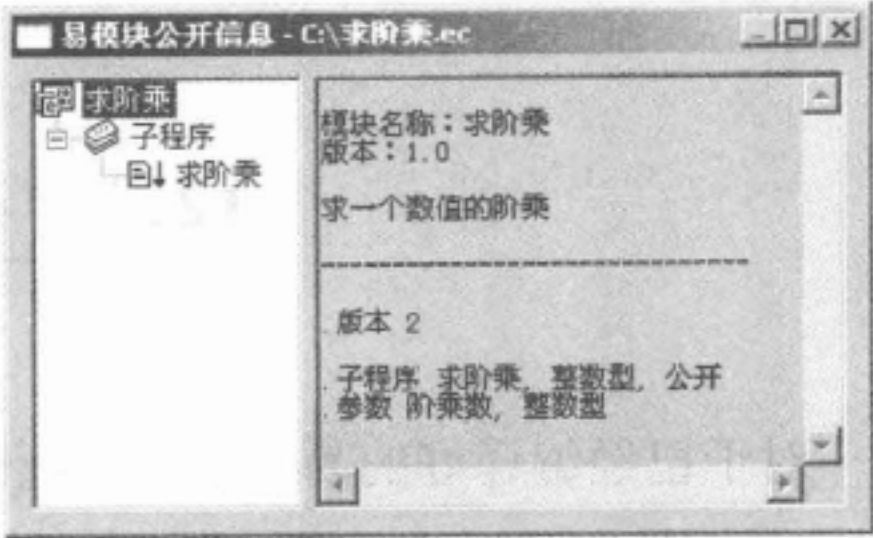


图12-10 “易模块公开信息”对话框

第三步：在易程序的启动窗口上添加标签组件、编辑框组件、按钮组件，界面如图12-11所示。

**注解：**模块导入后就可以直接调用模块中所有公开的资源和功能。  
双击“按钮1”，在“\_按钮1\_被单击”事件子程序中调用模块中所提供的子程序“求阶乘（）”，并添写其中的参数。编写如下代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

```
编辑框1.内容 = 到文本 (求阶乘 (到整数 (编辑框1.内容)))
```

按F5键运行程序。在“编辑框1”中输入整数5，点击按钮得到结果，如图12-12所示。

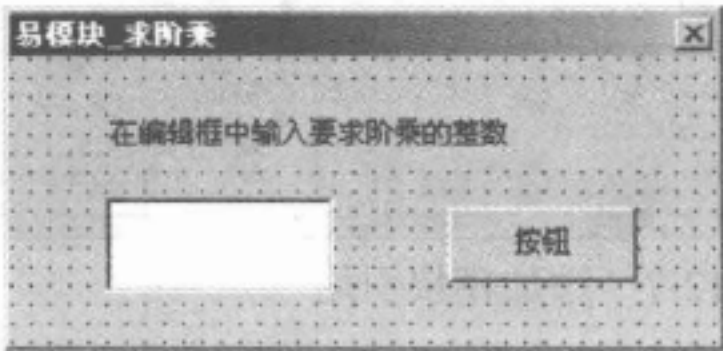


图12-11 在“\_启动窗口”添加的组件

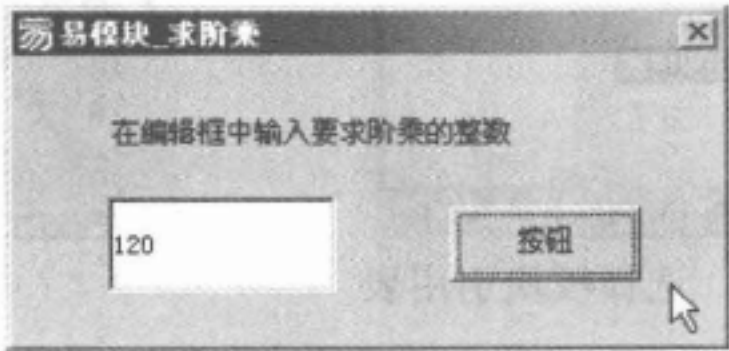


图12-12 运行效果

## 12.4 易模块的应用实例

在开发软件时，有时要求该软件同一时刻只能运行一个实例，不允许重复多次运行。根据这样的要求来编写禁止多次运行的模块，以供其他程序调用。

禁止多次运行程序的原理：要使软件同一时刻只能运行一个实例，将软件的运行看作是一个事件对象。在软件运行时，就根据事件对象的名称（该名称由用户自行定义的）使用API命令“打开事件对象”，判断是否有相同事件对象名称的事件对象在运行；如果有则返回真，如果没有则根据事件对象的名称使用API命令“创建事件对象”，在内存中创建事件对象，以备软件再运行时做判断使用；在软件关闭时，要使用API命令“关闭事件对象”销毁事件对象。



相关API命令的定义如下：

Dll命令名	返回值类型	公开	备 注	
创建事件对象	整数型			
Dll库文件名：				
kernel32				
在Dll库中对应命令名：				
CreateEventA				
参数名	类 型	传址	数组	备 注
安全特性	整数型			
重设事件	逻辑型			
触发状态	逻辑型			
对象名字	文本型			

- CreateEventA： 创建一个事件对象。CreateEventA参数说明见表12-1。

表12-1 CreateEventA参数说明

参 数	类型及说明
lpEventAttributes 安全特性	SECURITY_ATTRIBUTES，指定一个结构，用于设置对象的安全特性。如变成整数型，并传递零值，则表明使用对象默认的安全设置
bManualReset 重设事件	Long，如果为真，表示创建一个人工重设事件；如果为假，表示创建一个自动重设事件
bInitialState 触发状态	Long，如事件应内部进入触发状态，则为真
lpName 对象名字	String，指定事件对象的名字

Dll命令名	返回值类型	公开	备 注	
打开事件对象	整数型			
Dll库文件名：				
kernel32				
在Dll库中对应命令名：				
OpenEventA				
参数名	类 型	传址	数组	备 注
访问方法	整数型			
句柄继承	逻辑型			
对象名字	文本型			

- OpenEventA： 为一个已有命名的事件对象创建一个新句柄。OpenEventA参数说明见表12-2。





表12-2 OpenEventA参数说明

参 数	类型及说明
dwDesiredAccess 访问方法	Long, 下述常数之一： EVENT_ALL_ACCESS: 要求对事件对象进行完全访问 EVENT_MODIFY_STATE: 允许使用SetEvent 和 ResetEvent函数 SYNCHRONIZE: 允许事件对象的使用同步
bInheritHandle 句柄继承	Long, 句柄能够由子进程继承, 则为真
lpName 对象名称	String, 指定要打开的对象的名字

Dll命令名	返回值类型	公开	备 注	
关闭事件对象	整数型			
Dll库文件名：				
kernel32				
在Dll库中对应命令名：				
CloseHandle				
参数名	类 型	传址	数组	备 注
对象句柄	整数型			

- CloseHandle: 关闭一个内核对象。其中包括文件、文件映射、进程、线程、安全和同步对象等。CloseHandle参数说明见表12-3。

表12-3 CloseHandle参数说明

参 数	类型及说明
hObject 对象句柄	Long, 欲关闭的一个对象的句柄

第一步：新建一个易模块程序。在“程序集1”中创建“事件对象句柄”变量，用以保存创建的事件对象的句柄。

程序集名	保 留	保 留	备 注
程序集1			
变量名	类 型	数组	备 注
事件对象句柄	整数型		

再新建两个子程序。在这两个子程序中输入如下代码。

子程序名	返回值类型	公开	备 注		
程序是否已运行	逻辑型	✓	真表示已经运行，假表示还未运行		
参数名	类 型	参考	可空	数组	备 注
标志文本	文本型				



```

--- 如果真 (打开事件对象 (2031619, 假, 标志文本) ≠ 0)
    返回 (真)
↓
事件对象句柄 = 创建事件对象 (0, 假, 假, 标志文本)
返回 (假)
    
```

子程序名	返回值类型	公开	备 注
关闭事件对象		✓	窗口销毁时调用

关闭事件对象 (事件对象句柄)

第二步：根据传递的“标志文本”（即对象名称），判断该对象是否存在。

第三步：设置对事件对象进行完全访问（参数一为2031619），句柄不能够由子进程继承（参数二为假）。

```

打开事件对象 (2031619, 假, 标志文本)
    
```

参数一的值：

查询常量“EVENT\_ALL\_ACCESS”，得到（STANDARD\_RIGHTS\_REQUIRED+SYNCHRONIZE+3H）三个常量的和，再分别查询这三个常量，“STANDARD\_RIGHTS\_REQUIRED”为983040，“SYNCHRONIZE”为1048576，“3H”是常数3。将这三个整数相加得到2031619。

第四步：使用“标志文本”作为事件对象的名称，创建一个事件对象，返回事件对象的句柄。默认对象的安全设置（参数一为0），创建一个自动重设事件（参数二为假），事件进入外部触发状态（参数三为假）。

```

事件对象句柄 = 创建事件对象 (0, 假, 假, 标志文本)
    
```

在程序被关闭后，销毁对象句柄。

```

关闭事件对象 (事件对象句柄)
    
```

第五步：模块代码编写完后，选择“编译”菜单中“编译”选项，对此程序进行编译。保存为“禁止多次运行.ec”。

第六步：在另一个易语言程序中调用这个模块。新建一个易语言窗口程序，在“\_启动窗口”的事件中编写如下代码。

子程序名	返回值类型	公开	备 注
_启动窗口_创建完毕			

```

--- 如果真 (程序是否已运行 (“禁止多次运行”) = 真)
    信息框 (“程序已经运行！”, 0, )
    销毁 0
↓
    
```

子程序名	返回值类型	公开	备 注
_启动窗口_将被销毁			

关闭事件对象 0

将易语言窗口程序编译后运行。该程序只能运行一个实例，如图12-13所示。



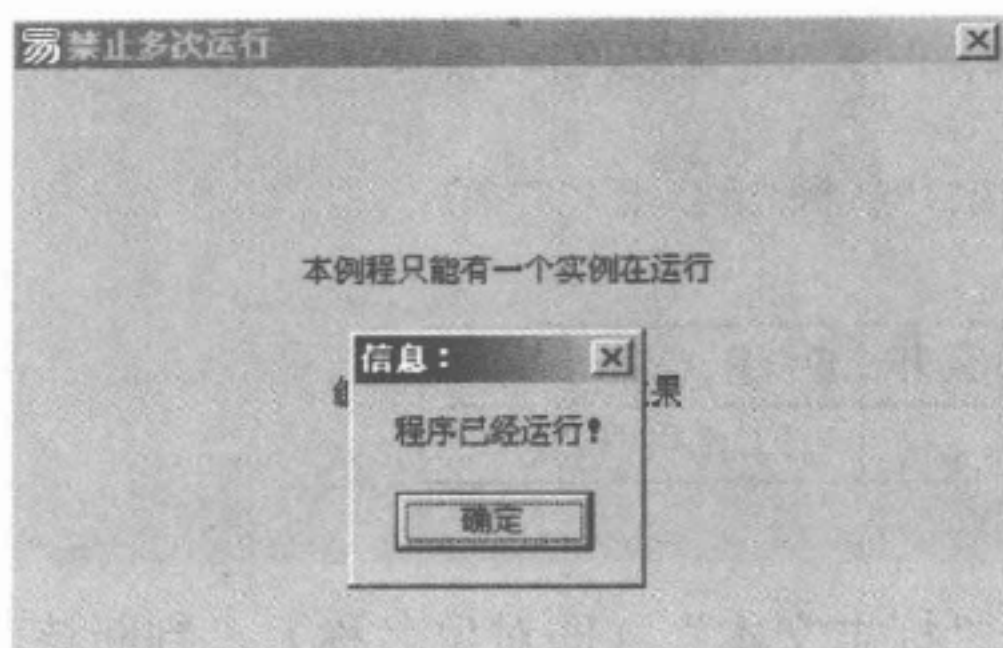


图12-13 禁止多次运行效果

至此，整个易模块的编写和调用就完成了。以后如果遇到需要限制多次运行的时候就不需要重新编写代码，直接导入禁止多次运行模块调用就可以了。

## 12.5 小结

模块是具有通用功能的命令的集合。易模块不用重复写程序代码，别人有的现成模块拿来即可。使用易模块可以再调用易模块，但嵌套深度建议小于三层。易语言编译时会将使用到的易模块代码一同编译到可执行文件中。

## 12.6 习题

- 12-1 深入了解易模块在程序模块化开发时所起的作用。
- 12-2 模块具有\_\_\_\_、\_\_\_\_、\_\_\_\_、\_\_\_\_基本属性。
- 12-3 试着编写一个求解一元二次方程的易模块。
- 12-4 试着编写一个十进制转换为二进制的易模块。
- 12-5 试着将其他程序中的子程序改写为易模块，并编程测试。





## 第十三章 易语言向导

### 本章目标

在本章结束时，我们能够：

- 了解什么是向导
- 了解易语言向导的意义
- 了解易语言向导支持库中的命令
- 掌握使用易语言向导程序
- 掌握创建易语言向导程序





## 13.1 了解易语言向导

向导是一种用来简化用户编写、开发、操作的程序，就是按照步骤引导用户一步步地进行某些操作，最后得到用户需求的结果。例如新建易语言程序也可以看作是一个向导程序，打开易语言选择菜单“程序→新建”可以选择多种程序设计方式，选择“Windows窗口程序”，易语言创建了一个主窗口供使用；选择“Windows动态链接库”，易语言创建了一个程序集，里面有“\_启动子程序”和“\_临时子程序”及简单说明；这都可以看做是向导程序运行的结果。

易语言向导是引导用户在新建的易程序或当前已打开的易程序中添加或插入自己需要的一些代码。易语言向导的功能就是提供一段完整的源程序代码，然后使用一个向导程序来引导用户一步步地在自己程序中导入或插入已经写好的源程序代码。

制作向导需要用户先写一个实现全部功能的完整源程序，该源程序就是一个模板程序（最后生成的代码就是通过对该模板进行编辑后的结果），然后编写一个向导界面，提供一些选项，最后根据最终的选项，对已经编写好的模板程序进行删除和编辑，最后将编辑好的代码插入到目的程序中。

使用易语言向导，可以方便代码的相互交流，以及方便他人根据需要使用自己的代码；还可以将自己经常使用的一些重复的代码编写成向导程序，当需要使用这些代码时，使用向导程序将代码插入到需要的程序中。

## 13.2 易语言向导支持库

易语言的向导制作需要使用易语言向导支持库提供的命令，该支持库中的命令都是针对易语言的源程序进行操作，这些命令可以动态地对源程序中的代码和组件等进行编辑，例如删除、复制、修改等操作，从而生成需要的代码。

通过查看支持库面板中易语言向导的命令与数据类型可以看出，易语言向导功能的强大，几乎可以对易程序进行各种操作。如对程序集、子程序、全局变量、程序集变量、局部变量、子程序参数、数据类型、DLL命令、窗口、窗口组件、常量或资源等进行任意的操作。

**注解：**易向导支持库中涉及到的“备注”沿用的是易语言3.7以前版本中的叫法，就是现在易语言代码中的注释。





易语言向导支持库具体涉及到的命令：

#### 1) “复制程序 ()” 命令

<无返回值> 复制程序 (程序项类型 欲复制项类型, 文本型 欲复制项名称, 文本型 新项目名称)

通过复制的方式建立指定程序项。

参数<1>的名称为“欲复制项类型”，类型为“程序项类型 (AppItemType)”。

参数<2>的名称为“欲复制项名称”，类型为“文本型 (text)”。

参数<3>的名称为“新项目名称”，类型为“文本型 (text)”。必须确保该名称合法且不重复。

例如：以“\_启动窗口”为模板，复制成“关于窗口”。

复制程序 (#程序项类型.窗口, “\_启动窗口”, “关于窗口”)

“\_启动窗口”上面的所有组件一并复制过去，这些组件在“关于窗口”上都能找到。

#### 2) “复制程序段 ()” 命令

<无返回值> 复制程序段 (文本型 子程序名称, 文本型 标记名, 文本型 复制到标记名)

通过复制的方式在指定子程序中建立一段程序代码，请一定注意确保欲复制的代码块完整，不交叉跨越分支及循环语句，否则将导致系统出错。

参数<1>的名称为“子程序名称”，类型为“文本型 (text)”。可以用“子程序名称”或“程序集名.子程序名”格式来表示。

参数<2>的名称为“标记名”，类型为“文本型 (text)”。标记可以在易语言程序中任何可以输入备注的地方设置，如程序语句、变量、窗口组件、自定义数据类型等，所设置标记在易语言向导最终所生成的程序中将被自动删除。标记的定义格式为“\$ (标记名称)”或“\$ (标记名称/)”加“\$ (/标记名称)”，后两个只能在程序语句中成对使用，用作定义具有多行语句的代码块标记，定义命令为在所选定程序块首语句的备注中加上“\$ (标记名称/)", 在所选定程序块末语句的备注中加上“\$ (/标记名称)", 两者名称必须相同。在程序语句备注中使用“\$ (标记名称)”格式设置标记，可定义该单行语句为一个代码块。同一个地方可以连续设置多个标记，譬如“\$ (标记名称1) \$ (标记名称2)”。标记可以与正常备注信息混合排放。

参数<3>的名称为“复制到标记名”，类型为“文本型 (text)”。本标记提供所选择程序段在本子程序内的目的复制位置。

例如：复制“\_\_启动窗口\_创建完毕”子程序的代码。

复制程序段 (“启动窗口程序集.\_\_启动窗口\_创建完毕”, “创建完毕标记”, “复制到标记”)



在“\_启动窗口\_创建完毕”子程序中，从标有“创建完毕标记”的代码开始复制到标有“复制到标记”的代码。

### 3) “删除程序 ()” 命令

<无返回值> 删除程序 (程序项类型 欲删除项类型, 文本型 欲删除项名称)

删除程序中的指定程序项。

参数<1>的名称为“欲删除项类型”，类型为“程序项类型 (AppItemType)”。

参数<2>的名称为“欲删除项名称”，类型为“文本型 (text)”。

例如：删除“关于窗口”的“加入按钮”组件。

删除程序 (#程序项类型.窗口组件, “关于窗口.加入按钮”)

“关于窗口”是由“\_启动窗口”复制来的，自然带有“\_启动窗口”上的所有组件，如果有不需要的组件应该将其删除。

### 4) “删除程序段 ()” 命令

<无返回值> 删除程序段 (文本型 子程序名称, 文本型 标记名)

删除指定子程序中一段程序代码，必须确保待删除的代码块完整，不交叉跨越分支及循环语句，否则将导致系统出错。

参数<1>的名称为“子程序名称”，类型为“文本型 (text)”。可以用“子程序名称”或“程序集名.子程序名”格式来表示。

参数<2>的名称为“标记名”，类型为“文本型 (text)”。标记可以在易语言程序中任何可以输入备注的地方设置，如程序语句、变量、窗口组件、自定义数据类型等，所设置标记在易语言向导最终生成的程序中将被自动删除。标记的定义格式为“\$ (标记名称)”或“\$ (标记名称/)”加“\$ (/标记名称)”，后两个只能在程序语句中成对使用，用作定义具有多行语句的代码块标记，定义命令为在所选定程序块首语句的备注中加上“\$ (标记名称/)", 在所选定程序块末语句的备注中加上“\$ (/标记名称)”，两者名称必须相同。在程序语句备注中使用“\$ (标记名称)”格式设置标记，可定义该单行语句为一个代码块。同一个地方可以连续设置多个标记，如“\$ (标记名称1) \$ (标记名称2)”。标记可以与正常备注信息混合排放。

例如：删除“\_启动窗口\_创建完毕”子程序中一段做了“代码标记”标记对的代码。

删除程序段 (“启动窗口程序集.\_启动窗口\_创建完毕”, “代码标记”)

### 5) “删除标记程序 ()” 命令

<无返回值> 删除标记程序 (文本型 标记名)

删除程序中所有注释内包含指定标记的各类程序项目 (不包括程序代码段)。

参数<1>的名称为“标记名”，类型为“文本型 (text)”。标记可以在易语言程序中任何可以输入备注的地方设置，如程序语句、变量、窗口组件、自定义数据类型等，





所设置标记在易语言向导最终生成的程序中将被自动删除。标记的定义格式为“\$（标记名称）”或“\$（标记名称/）”加“\$（/标记名称）”，后两个只能在程序语句中成对使用，用作定义具有多行语句的代码块标记，定义命令为在所选定程序块首语句的备注中加上“\$（标记名称/）”，在所选定程序块末语句的备注中加上“\$（/标记名称）”，两者名称必须相同。在程序语句备注中使用“\$（标记名称）”格式设置标记，可定义该单行语句为一个代码块。同一个地方可以连续设置多个标记，如“\$（标记名称1）\$（标记名称2）”。标记可以与正常备注信息混合排放。

例如：删除做了“代码删除”标记的代码所在行。

删除标记程序（“代码删除”）

#### 6) “修改程序（）”命令

<无返回值> 修改程序（程序项类型 欲修改项类型，文本型 欲修改项名称，程序项属性 欲修改属性，通用型 修改值）

修改程序中的指定程序项。

参数<1>的名称为“欲修改项类型”，类型为“程序项类型（AppItemType）”。

参数<2>的名称为“欲修改项名称”，类型为“文本型（text）”。

参数<3>的名称为“欲修改属性”，类型为“程序项属性（AppItemProperty）”。

合法的修改属性与修改项类型相关，具体如下。

程序集：名称、备注、公开。

子程序：名称、备注、数据类型、公开、收缩。

全局变量、程序集变量、局部变量、子程序参数、数据类型成员、DLL命令参数（名称、备注、数据类型、数组类型、静态、参考、可空、公开）。

数据类型：名称、备注、公开。

DLL命令：名称、备注、公开、DLL库文件名、DLL库命令名、数据类型。

窗口：名称、备注。

窗口组件：名称、备注。

常量或资源：名称、备注、公开、常量或资源值。

参数<4>的名称为“修改值”，类型为“通用型（all）”。修改值的数据类型必须与指定欲修改属性的数据类型一致，修改值具体所需要的数据类型请见“程序项属性”中各成员常量的解释。

例如：修改“关于窗口”组件的备注为“关于窗口”。

修改程序（#程序项类型.窗口，“关于窗口”，#程序项属性.备注，“关于窗口”）

#### 7) “置组件属性（）”命令

<无返回值> 置组件属性（文本型 组件属性名，通用型 属性值）

设置组件属性。





参数<1>的名称为“组件属性名”，类型为“文本型(text)”。组件属性名以“窗口名.组件名.属性名”或“窗口名.属性名”格式表示，注意窗口组件包括菜单项。

参数<2>的名称为“属性值”，类型为“通用型(all)”。属性值的数据类型必须与指定组件属性的数据类型一致。

例如：设置“关于窗口”的标题为“关于”。

置组件属性(“关于窗口.标题”，“关于”)

#### 8) “置语句备注 ()” 命令

<无返回值> 置语句备注 (文本型 子程序名称, 文本型 标记名, [文本型 备注])

设置子程序中指定语句的备注信息。

参数<1>的名称为“子程序名称”，类型为“文本型(text)”。可以用“子程序名称”或“程序集名.子程序名”格式来表示。

参数<2>的名称为“标记名”，类型为“文本型(text)”。标记可以在易语言程序中任何可以输入备注的地方设置，如程序语句、变量、窗口组件、自定义数据类型等，所设置标记在易语言向导最终生成的程序中将被自动删除。标记的定义格式为“\$(标记名称)”或“\$(标记名称/)”加“\$(/标记名称)”，后两个只能在程序语句中成对使用，用作定义具有多行语句的代码块标记，定义命令为在所选定程序块首语句的备注中加上“\$(标记名称/)", 在所选定程序块末语句的备注中加上“\$(/标记名称)”，两者名称必须相同。在程序语句备注中使用“\$(标记名称)”格式设置标记，可定义该单行语句为一个代码块。同一个地方可以连续设置多个标记，如“\$(标记名称1)\$ (标记名称2)”。标记可以与正常备注信息混合排放。

参数<3>的名称为“备注”，类型为“文本型(text)”，可以被省略。如果本参数被省略，默认值为空文本。

例如：设置标有“代码标记”标记的代码的注释。

置语句备注(“\_\_启动窗口\_创建完毕”，“代码标记”，“备注信息”)

#### 9) “置程序信息 ()” 命令

<无返回值> 置程序信息 ([文本型 程序名称], [整数型 程序类型], [文本型 备注], [文本型 版本], [字节集 图标])

设置程序整体的相关信息。

参数<1>的名称为“程序名称”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<2>的名称为“程序类型”，类型为“整数型(int)”，可以被省略。具体值为：1 (Windows窗口程序)；2 (Windows控制台程序)；3 (Windows动态链接库)；4 (Windows易语言模块)；5 (Linux控制台程序)；6 (Linux易语言模块)。如果本参数被省略，则不改动此项。





参数<3>的名称为“备注”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<4>的名称为“版本”，类型为“文本型(text)”，可以被省略。版本文本格式为“主版本号.次版本号”，如果本参数被省略，则不改动此项。

参数<5>的名称为“图标”，类型为“字节集(bin)”，可以被省略。如果本参数被省略，则不改动此项。

例如：设置程序的配置信息，可在菜单“程序”→“配置”项中查看到设置的信息。

置程序信息(“程序名称”,1,“备注”,“1.0”,#图标)

#### 10) “置作者信息 ()” 命令

<无返回值> 置作者信息 ([文本型 作者名称], [文本型 邮政编码], [文本型 联系地址], [文本型 电话], [文本型 传真], [文本型 电子信箱], [文本型 主页地址], [文本型 其他信息])

设置程序与作者相关的信息。可在菜单“程序”→“配置”项中查看到设置的信息。

参数<1>的名称为“作者名称”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<2>的名称为“邮政编码”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<3>的名称为“联系地址”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<4>的名称为“电话”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<5>的名称为“传真”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<6>的名称为“电子信箱”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<7>的名称为“主页地址”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

参数<8>的名称为“其他信息”，类型为“文本型(text)”，可以被省略。如果本参数被省略，则不改动此项。

#### 11) “定义模板变量 ()” 命令

<无返回值> 定义模板变量 (文本型 模板变量名称, [通用型 模板变量值])

设置模板变量的值，模板变量可以用作自动程序替换。可以在模板易语言程序中使用模板变量与模板变量条件表达式。

参数<1>的名称为“模板变量名称”，类型为“文本型(text)”。





参数<2>的名称为“模板变量值”，类型为“通用型（all）”，可以被省略。只能是系统基本数据类型数据。如果本参数被省略，默认值为空文本。

例如：定义名称为“配置文件保存”的模板变量。

定义模板变量（“配置文件保存”，）

#### 12) “删除模板变量（）”命令

<无返回值> 删除模板变量（文本型 模板变量名称）

删除先前所定义的模板变量。

参数<1>的名称为“模板变量名称”，类型为“文本型（text）”。

例如：删除模板变量“配置文件变量”。

删除模板变量（“配置文件变量”）

如果删除了定义的模板变量“配置文件变量”，那么该模板变量要替换的内容将不会被写出。

#### 13) “清除修改记录（）”命令

<无返回值> 清除修改记录（）

清除先前所有修改或设置命令所记录下来的模板程序修改记录。

#### 14) “写出程序（）”命令

<文本型> 写出程序（字节集 模板程序数据，[逻辑型 是否关闭源程序]）

根据已有的模板程序修改记录对模板程序进行实际修改。如果修改成功返回空文本，否则返回具体的错误信息。修改成功后的结果在向导程序退出后将被易系统自动载入编辑。无论写出成功或者失败，本命令退出时都会自动清除所有模板程序修改信息。

参数<1>的名称为“模板程序数据”，类型为“字节集（bin）”。此数据为用作模板程序的完整易源程序。

参数<2>的名称为“是否关闭源程序”，类型为“逻辑型（bool）”，可以被省略。如果本参数为真，则当执行写出程序时，先将现行易语言编辑环境中的源程序关闭，然后将所写出程序作为新的完整程序载入。如果本参数为假，当执行写出程序时，会将所生成程序与现行易语言编辑环境中的源程序合并（合并前“\_启动窗口”及其窗口程序集、“启动子程序”代码将被抛弃）。如果本参数被省略，默认值为真。

例如：将编辑完的向导代码，写到程序中。

写出程序（#向导模板程序，）

#### 15) “添加模块引用（）”命令

<无返回值> 添加模块引用（文本型 模块文件路径名）

增加对指定易模块文件的引用到程序中。

参数<1>的名称为“模块文件路径名”，类型为“文本型（text）”。本参数提供





欲添加引用的模块文件路径名。如果以“\$”字符开始，表示后面是基于易语言系统模块目录的相对路径。如：\$classlib\d3d.ec，代表在易语言系统模块目录下classlib子目录中的d3d.ec模块文件。

例如：添加易语言系统模块目录中的取汉字笔画模块。

添加模块引用 (“\$取汉字笔画.ec” )

16) “删除模块引用 ( )” 命令

<无返回值> 删除模块引用 (文本型 模块文件名)

删除程序中对指定易模块文件的引用。

参数<1>的名称为“模块文件名”，类型为“文本型 (text)”。本参数提供欲删除引用的不带路径易模块文件名。

例如：删除取汉字笔画模块。

删除模块引用 (“取汉字笔画.ec” )

对易语言向导支持库命令的使用也许有些抽象和模糊，下面具体来看一个编辑向导代码的例子，以加深对易语言向导支持库命令的了解。

第一步：新建一个易程序，在“启动窗口”上添加一个“编辑框1”组件，将“编辑框1”的“是否允许多行”属性设为真。双击“启动窗口”，在“\_启动窗口\_创建完毕”子程序中输入向导模板程序的代码，如下所示：

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

```

编辑框1.加入文本 (“启动窗口创建完毕”，#换行符)
' $ (创建完毕标记)
编辑框1.加入文本 (“复制标记代码”，#换行符)
' $ (复制到标记)
' $如果 (判断变量=0)
编辑框1.加入文本 (“模板判断变量=0”)
' $否则
' $如果 (判断变量<0)
编辑框1.加入文本 (“模板判断变量≠0”)
' $结束
' $如果 (删除变量=“删除”)
编辑框1.加入文本 (“删除模板变量”)
' $结束
    
```

第二步：将编写好的代码保存成“代码向导模板.e”文件。再新建一个易程序，在“资源”中加入保存的“代码向导模板.e”文件。

图片或图片组名称	内容	公开	备注
代码向导	2590		





第三步：双击“启动窗口”，在“\_启动窗口\_创建完毕”子程序中输入向导编辑程序，代码如下：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

复制程序段（“启动窗口程序集.\_启动窗口\_创建完毕”，“创建完毕标记”，“复制到标记”）

置语句备注（“启动窗口程序集.\_启动窗口\_创建完毕”，“创建完毕标记”，“当启动窗口创建完毕后，向编辑框中加入文本”）

定义模板变量（“判断变量”，1）

定义模板变量（“删除变量”，“删除”）

删除模板变量（“删除变量”）

写出程序（#代码向导，真）

销毁（）

第四步：将编写好的代码保存成“代码向导.e”文件并编译成“代码向导.exe”文件。执行此向导程序，执行后的结果如下：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

编辑框1.加入文本（“启动窗口创建完毕”，#换行符）

当启动窗口创建完毕后，向编辑框中加入文本

编辑框1.加入文本（“复制标记代码”，#换行符）

编辑框1.加入文本（“模板判断变量≠0”）

向导代码对比解释：向导执行的是“代码向导.exe”文件。

（1）向导执行后，没有对此行代码做任何处理，直接在新程序加入此行代码。

编辑框1.加入文本（“启动窗口创建完毕”，#换行符）

（2）根据标记文本，使用“复制程序段”命令，将标记文本标记的程序或程序段加入到新程序中。

复制程序段（“启动窗口程序集.\_启动窗口\_创建完毕”，“创建完毕标记”，“复制到标记”）

对应的模板代码如下：

\$（创建完毕标记）

编辑框1.加入文本（“复制标记代码”，#换行符）

\$（复制到标记）

（3）根据标记文本，使用“置语句备注”命令，在标记文本所在位置加入代码语句的注释信息。

（4）根据定义的模板变量“判断变量”，使用“定义模板变量”命令，对“判断变量”进行赋值，在向导程序中对所赋的值进行判断，将判断后要执行的代码加入到新程序中。

定义模板变量（“判断变量”，1）





```

' $如果 (判断变量=0)
编辑框1.加入文本 ("模板判断变量=0")
' $否则
' $如果 (判断变量<>0)
编辑框1.加入文本 ("模板判断变量≠0")
' $结束

```

(5) 判断模板变量“删除变量”的值，因为使用了“删除模板变量”命令删除了“删除变量”，所以代码不会写到新程序中。

定义模板变量(“删除变量”，“删除”)

删除模板变量(“删除变量”)

对应的模板代码如下：

```

' $如果 (删除变量="删除")
编辑框1.加入文本 ("删除模板变量")
' $结束

```

(6) 写出程序，销毁启动窗口。

向导程序中代码的编辑基本就是这样的流程，下面来了解向导程序中组件的编辑情况。

### 13.3 易语言向导的编写

在深入了解了易语言向导支持库的命令后，再来编写一个自动生成菜单的易语言向导程序。编写易语言向导程序大体分为两个步骤，首先是根据软件的需要，完成模板程序的所有代码；其次，就是编写编辑模板程序的代码。

第一步：编写模板程序。新建一个易程序，建立模板程序的菜单，如图13-1所示。

① 因为要编写的是自动生成菜单的向导程序，所以不需要加任何代码。编写完模板程序，将其保存，再来编写向导例程。

② 新建一个易程序，在“启动窗口”中加入两个按钮，一个“取消按钮”，用作取消向导程序；一个“完成按钮”，用作生成向导程序。加

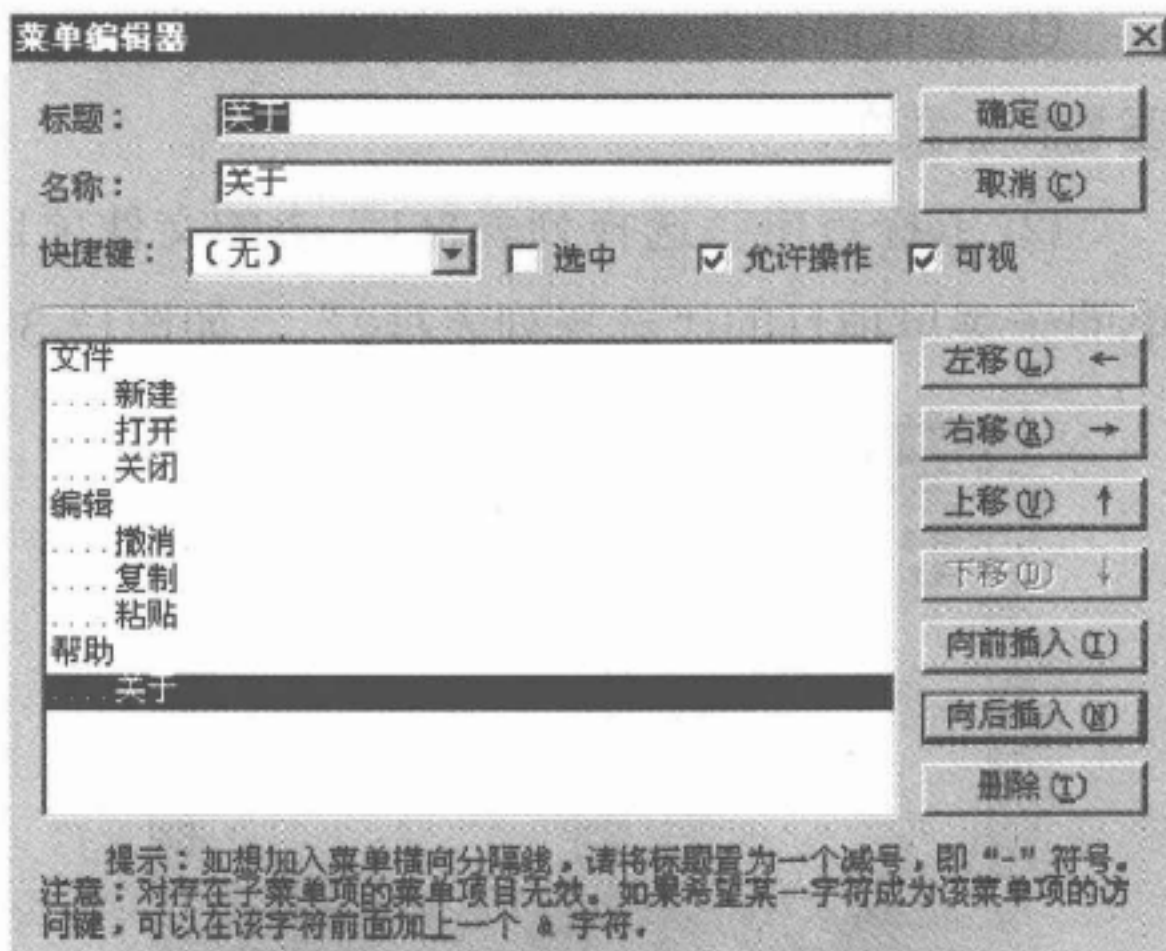


图13-1 模板程序菜单





入四个选择列表框，用作显示供选择的菜单项目。“选择列表框1”显示顶级菜单“文件”、“编辑”、“帮助”的菜单名称；“选择列表框2”显示“文件”菜单的子菜单“新建”、“打开”、“关闭”的子菜单名称；“选择列表框3”显示“编辑”菜单的子菜单“撤消”、“复制”、“粘贴”的子菜单名称；“选择列表框4”显示“帮助”菜单的子菜单“关于”的子菜单名称。

③ 将四个选择列表框的尺寸设为一样的；将“选择列表框2”、“选择列表框3”、“选择列表框4”的位置设为一样的；“选择列表框3”的可视属性设为假，“选择列表框4”的可视属性设为假；调整“选择列表框2”的层次为顶层。

④ 程序启动后当点击“选择列表框1”中的“文件”项目时，在程序中使用代码将“选择列表框2”的可视属性置为真，“选择列表框3”和“选择列表框4”的可视属性置为假，就可以看到“选择列表框2”，并可以对“选择列表框2”的项目进行选择了；当点击“选择列表框1”中的“编辑”项目时，在程序中使用代码将“选择列表框3”的可视属性置为真，“选择列表框2”和“选择列表框4”的可视属性置为假，就可以看到“选择列表框3”，并可以对“选择列表框3”的项目进行选择了；当点击“选择列表框1”中的“帮助”项目时，在程序中使用代码将“选择列表框4”的可视属性置为真，“选择列表框2”和“选择列表框3”的可视属性置为假，就可以看到“选择列表框4”，并可以对“选择列表框4”的项目进行选择了。

设计完的界面如图13-2所示。

将保存的模板程序当作资源加入到向导例程的资源表中。如下所示：

图片或图片组名称	内容	公开	备注
向导程序	2579		

第二步：编写对模板进行修改的代码。

① 在代码中要判断三个菜单及其所属菜单项的选择情况，根据选择情况，决定保留还是删除菜单。

② 鼠标点击“选择列表框1”中的文件项目，程序窗口的右边显示包含了“文件”菜单的子菜单项目的“选择列表框2”，如图13-3所示。

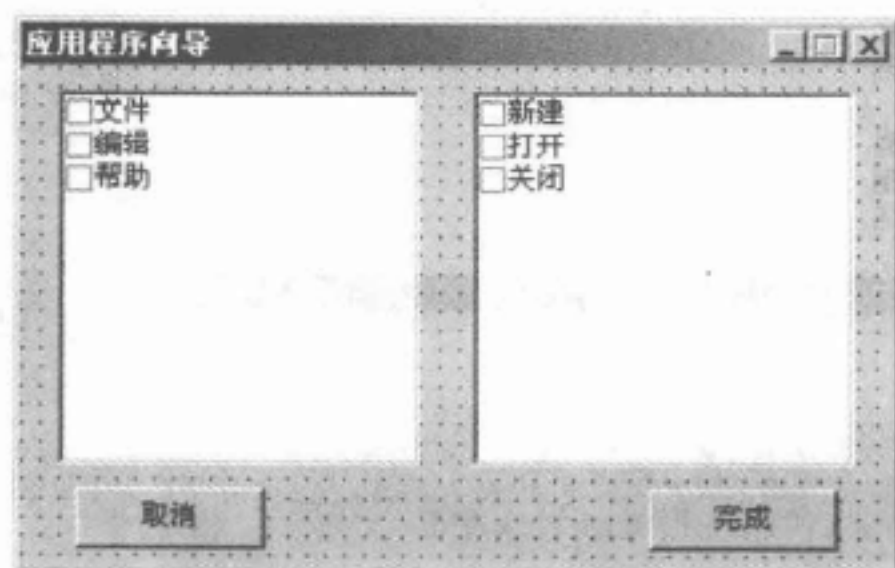


图13-2 向导程序界面

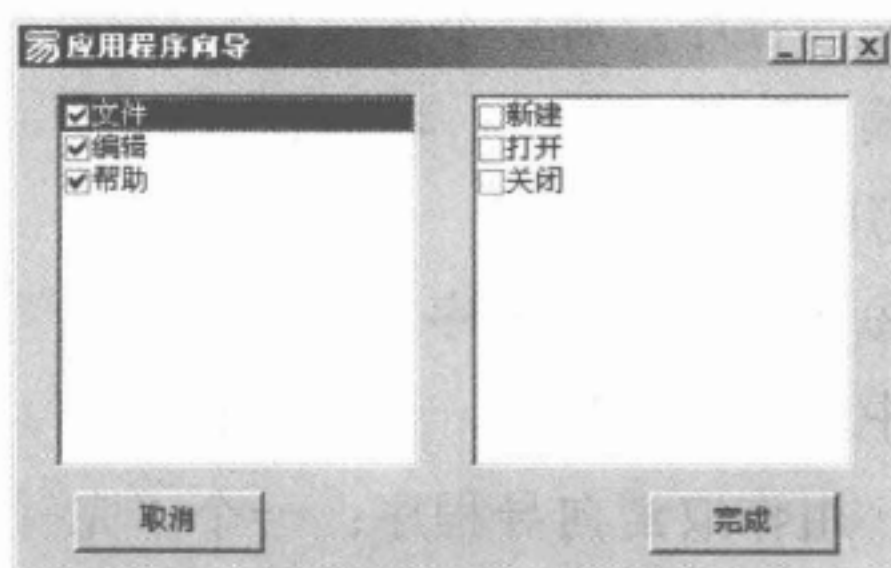


图13-3 选择文件菜单





③ 对于“文件”菜单，要先判断选择列表框1中的“文件”项目是否被选中，如果没有被选中，就将“文件”菜单及其对应的子菜单项全部删除；如果选择列表框1中的“文件”项目被选中，就再判断“选择列表框2”中“文件”菜单所对应的子菜单项是否被选中，哪一项子菜单项没有被选中就删除哪一项子菜单项，如果“选择列表框2”中“文件”菜单所对应的子菜单项都没有被选中，那么就将“文件”菜单及其所对应的子菜单项全部删除。代码如下：

```

--- 判断 (选择列表框1.是否被选中 (0) = 真) ^ 文件菜单
--- 如果真 (选择列表框2.是否被选中 (0) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.新建”)
--- 如果真 (选择列表框2.是否被选中 (1) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.打开”)
--- 如果真 (选择列表框2.是否被选中 (2) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.关闭”)
--- 如果真 (选择列表框2.是否被选中 (0) = 假 且 选择列表框2.是否被选中 (1) = 假
    且 选择列表框2.是否被选中 (2) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.文件”)
--- 删除程序 (#程序项类型.窗口组件, “_启动窗口.文件”)
--- 删除程序 (#程序项类型.窗口组件, “_启动窗口.新建”)
--- 删除程序 (#程序项类型.窗口组件, “_启动窗口.打开”)
--- 删除程序 (#程序项类型.窗口组件, “_启动窗口.关闭”)

```

④ 鼠标点击“选择列表框1”中的编辑项目，程序窗口的右边显示包含了“编辑”菜单的子菜单项目的“选择列表框3”，如图13-4所示。

⑤ 对于“编辑”菜单，要先判断选择列表框1中的“编辑”项目是否被选中，如果没有被选中，就将“编辑”菜单及其对应的子菜单项全部删除；如果选择列表框1中的“编辑”项目被选中，就再

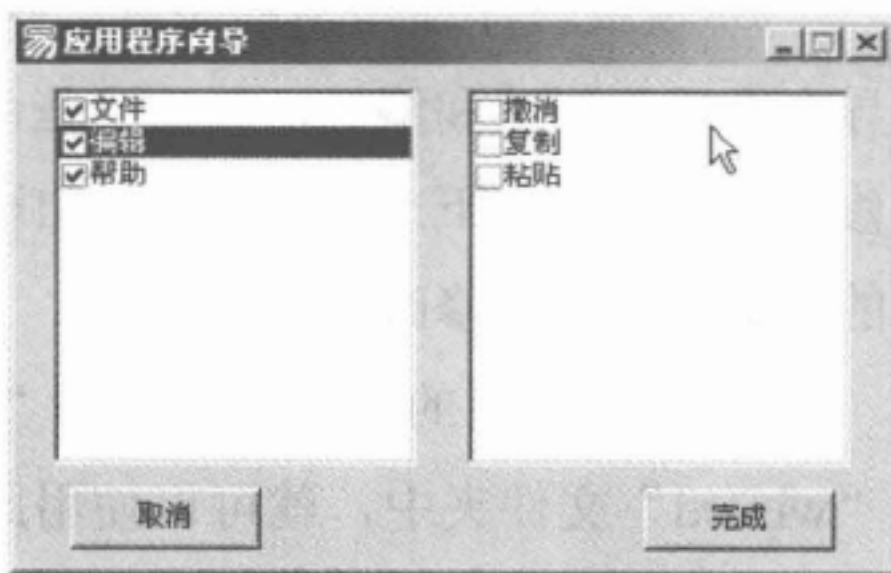


图13-4 选择编辑菜单

判断“选择列表框3”中“编辑”菜单所对应的子菜单项是否被选中，哪一项子菜单项没有被选中就删除哪一项子菜单项，如果“选择列表框3”中“编辑”菜单所对应的子菜单项都没有被选中，那么就将“编辑”菜单及其所对应的子菜单项全部删除。代码如下：

```

--- 判断 (选择列表框1.是否被选中 (1) = 真) ^ 编辑菜单
--- 如果真 (选择列表框3.是否被选中 (0) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.撤销”)
--- 如果真 (选择列表框3.是否被选中 (1) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.复制”)
--- 如果真 (选择列表框3.是否被选中 (2) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.粘贴”)
--- 如果真 (选择列表框3.是否被选中 (0) = 假 且 选择列表框3.是否被选中 (1) = 假
    且 选择列表框3.是否被选中 (2) = 假)
    删除程序 (#程序项类型.窗口组件, “_启动窗口.编辑”)

```





```

-- 且 选择列表框3.是否被选中 (2) = 假)
删除程序 (#程序项类型.窗口组件, "_启动窗口.编辑")

```

```

-- 删除程序 (#程序项类型.窗口组件, "_启动窗口.编辑")
-- 删除程序 (#程序项类型.窗口组件, "_启动窗口.撤消")
-- 删除程序 (#程序项类型.窗口组件, "_启动窗口.复制")
-- 删除程序 (#程序项类型.窗口组件, "_启动窗口.粘贴")

```

⑥ 判断完“文件”菜单和“编辑”菜单，再依例判断“帮助”菜单，方法与判断“文件”菜单和“编辑”菜单一样。

⑦ 当所有对模板程序的修改代码完成后，最后用“写出程序（）”命令，将修改后的程序代码写到新建的易程序中。将写出程序的结果放到“结果文本”变量中，判断“结果文本”是否为空就知道向导程序是否创建成功了。

向导最终写出程序的代码如下：

```

结果文本 = 写出程序 (#向导程序, ) ' 将程序作为新的完整程序载入
-- 判断 (结果文本 = "")
-- 信息框 ("已经创建 应用程序", 0, )
-- 信息框 (结果文本, 0, )
销毁 0

```

⑧ 模板程序在写出程序前，并不需要与向导程序做关联，大家只需要使用易语言向导命令进行修改和编辑，其实这些命令使用时，并没有直接对模板文件进行修改，而是将修改的步骤记录下来，使用“写出程序（）”命令时，将模板文件读到内存中，对内存中的模板文件进行修改。

⑨ 将编译后的向导程序（如“易语言向导例程.exe”）复制到易语言安装目录下的“wizard”文件夹中，就可以使用此向导了。

**注解：**（1）在向当前打开易程序中插入代码时，插入代码中的程序项目和当前打开易程序中程序项重名时，插入的代码的程序项会自动更名，所以要注意不要使用“\_启动窗口”等常用的名称。例如模板程序中有“\_启动窗口”，当前打开的易程序中也有“\_启动窗口”，则插入到当前打开易程序中的“\_启动窗口”会自动更名。

（2）易语言向导支持库中很多命令都有“程序项名称”，这些参数都要使用“父项目.子项目”的形式表示。例如，“删除程序（）”、“复制程序（）”命令。

（3）如果要制作可以在当前打开易程序中插入代码的易语言向导，要将“写出程序（）”命令的第二个参数设置为“假”，并且要注意程序项名称不要与“\_启动窗口”等经常使用的程序项重名。

（4）模板程序代码中的条件语句最好写在空行的注释中。





## 13.4 易语言向导的使用方法

完成了一个向导程序，下面来看看怎么使用易语言向导。

易语言向导程序是一个非独立编译的易程序，该程序存放在指定的文件夹中（易语言安装目录下的“wizard”文件夹）。

第一步：选择“程序→新建”，选中“通过向导创建”→“常用向导”，就可以看到易语言安装目录下的“wizard”文件夹里的所有向导程序，如图13-5所示。

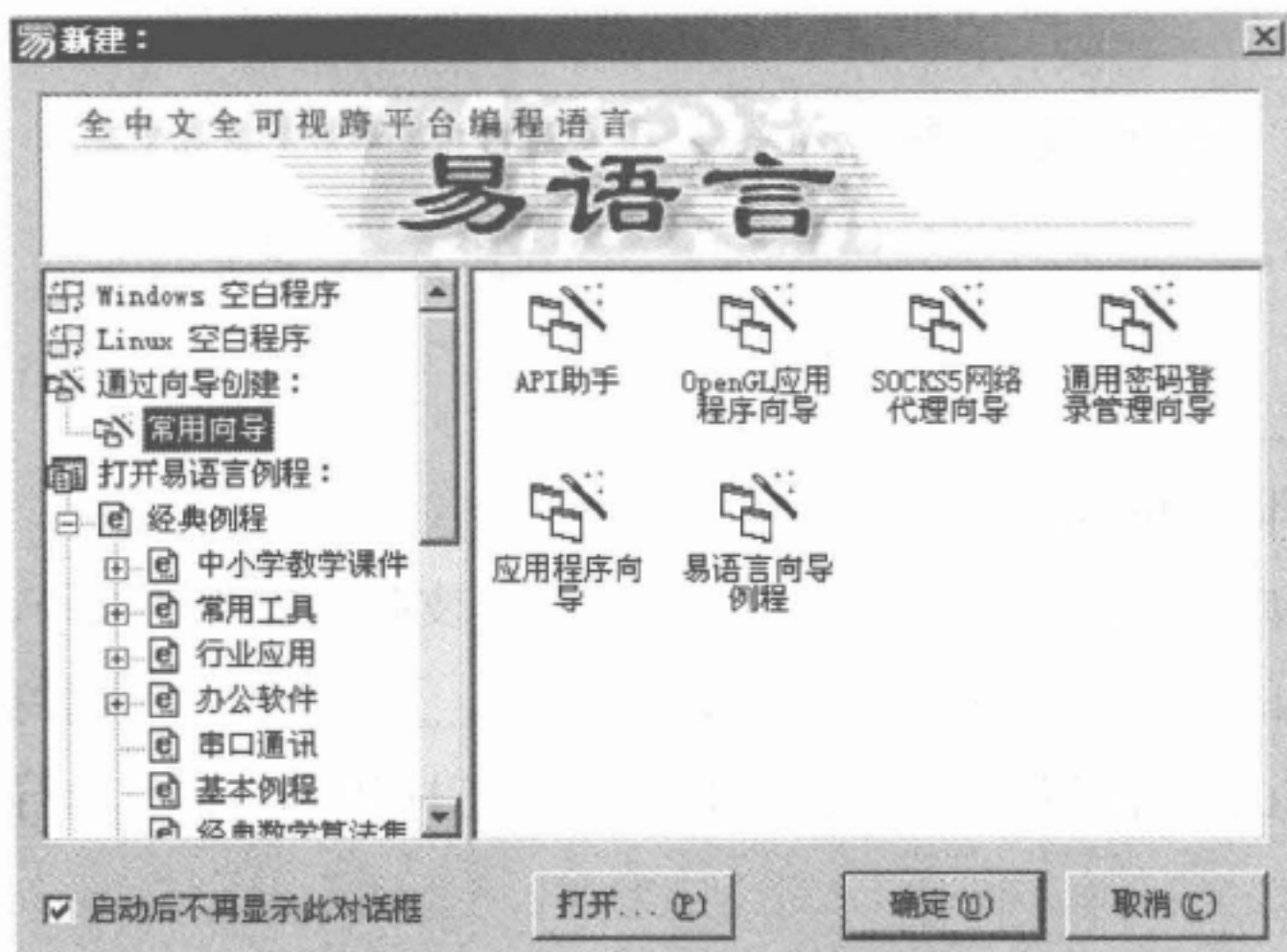


图13-5 新建易程序对话框中使用易语言向导

第二步：选择“易语言向导例程”，双击运行。选择需要的菜单项后点击“完成”按钮，完成向导程序的运行，生成的程序会自动带有已选择的菜单项，如图13-6所示。

易语言向导除了可以在“新建易程序对话框”中运行外，还可以通过菜单：“工具→执行易语言向导”来运行。通过该方法执行的易语言向导，可以在当前已打开的易程序中插入代码。这里要注意，如果是编写可以在已打开的易程序中插入代码的向导，在写易语言向导代码时，还要对“写出程序（）”命令的参数进行设置。

所以，易语言向导程序必须在“新建易程序对话框”和工具菜单中的“执行易语言向导”中进行调用，任何易语言向导程序脱离易语言单独运行，都是没有效果的。这是因为向导程序要和易语言程序要进行通信，易语言需要给易语言向导程序提供写出代码的接口。



图13-6 向导程序创建成功





## 13.5 小结

向导是一个应用程序。向导程序就是一个完整的源程序。创建向导程序就是从源程序中选择需要的功能创建出新的程序，编辑时要注意标记文本的对称和模板变量的定义。其实，易语言的向导程序就是将易程序的源代码做为一个模板资源，读入到内存中，在向导程序中使用易语言向导支持库的命令，对模板源代码和组件进行删除、修改，最后将剩余的源代码和组件写到新的易程序中。

## 13.6 习题

- 13-1 向导是\_\_\_\_\_。
- 13-2 易语言向导的功能是\_\_\_\_\_。
- 13-3 易语言向导的制作需要使用\_\_\_\_\_支持库提供的命令。
- 13-4 编译后的易语言向导程序应复制到易语言\_\_\_\_\_文件夹下。
- 13-5 在当前已打开的易程序中插入代码，要怎样执行易语言向导？







## 第十四章 程序调试

### 本章目标

在本章结束时，我们能够：

- 了解什么是程序调试
- 了解程序调试的意义
- 掌握如何使用“运行”菜单中提供的调试功能
- 掌握调试中跟踪查改变量的方法
- 掌握如何使用核心支持库提供的调试命令



## 14.1 了解程序调试

所谓程序调试，是在开发的程序投入实际使用前，用手工或编译程序等方法进行测试，修正语法错误和逻辑错误的过程。这是保证程序正确执行必不可少的步骤。

每一个程序几乎都需要经过反复的调试、修改，才能最终完成。当发现编写的程序没有正确地按预期执行，说明程序中某个地方出现了错误。很显然，只有找出错误的地方才可以将其修正。本部分将介绍如何快速、准确地找到错误的地方。

程序中出现的错误一般分为“语法错误”和“逻辑错误”。

“语法错误”指程序代码不符合易语言的语法规则。这种错误最容易发现并修改：首先在代码输入的时候，易语言系统会检查并发现一部分语法错误；其次在程序运行的时候，程序执行到有语法错误的代码行，光标停止在错误所在行并在输出面板中指出错误原因。由此可见，易语言系统会自动找出绝大部分的语法错误，只要按照其提示信息进行相应的修改即可。

“逻辑错误”是指程序流程上、数据处理上的错误。含有“逻辑错误”的程序能够正常执行，只是执行结果与程序最终要完成的结果不一致。这类错误易语言系统是不会发现的，只有编程者自己去查找。实际应用中，通常有“断点”、“跟踪”、“查改变量”等方法，实际中常将其结合起来使用，以确定出问题的地方。

## 14.2 运行调试

下面将介绍易语言中调试程序的具体方法。程序调试的一般步骤如下：

第一步：编写代码时，每输入一行代码，易语言系统就会对输入的代码进行语法检查。

第二步：执行所有的程序功能，从中找出并修改易语言系统提示的“语法错误”。

第三步：如果调试运行后，程序结果与预想的不一致，通过分析判断，找到可能有“逻辑错误”的代码段，在前面设置“断点”。

第四步：待程序在“断点”处中断后，使用“跟踪”、“查改变量”等方法，检查运行过程中每一步是否执行正确，变量的数据是否有误。

第五步：找到错误后，终止程序运行，修改代码。继续调试，直到程序所有功能全部正常为止。

### 14.2.1 预编译调试

每输入一行程序代码，换行时易语言都会对这一行代码进行预编译调试。如果发现有





明显的语法错误存在，易语言会将光标自动移动到错误行，并在“状态夹”的“输出”面板中提示错误原因，用户可以根据光标提示和错误原因很快改正错误，如图14-1所示。

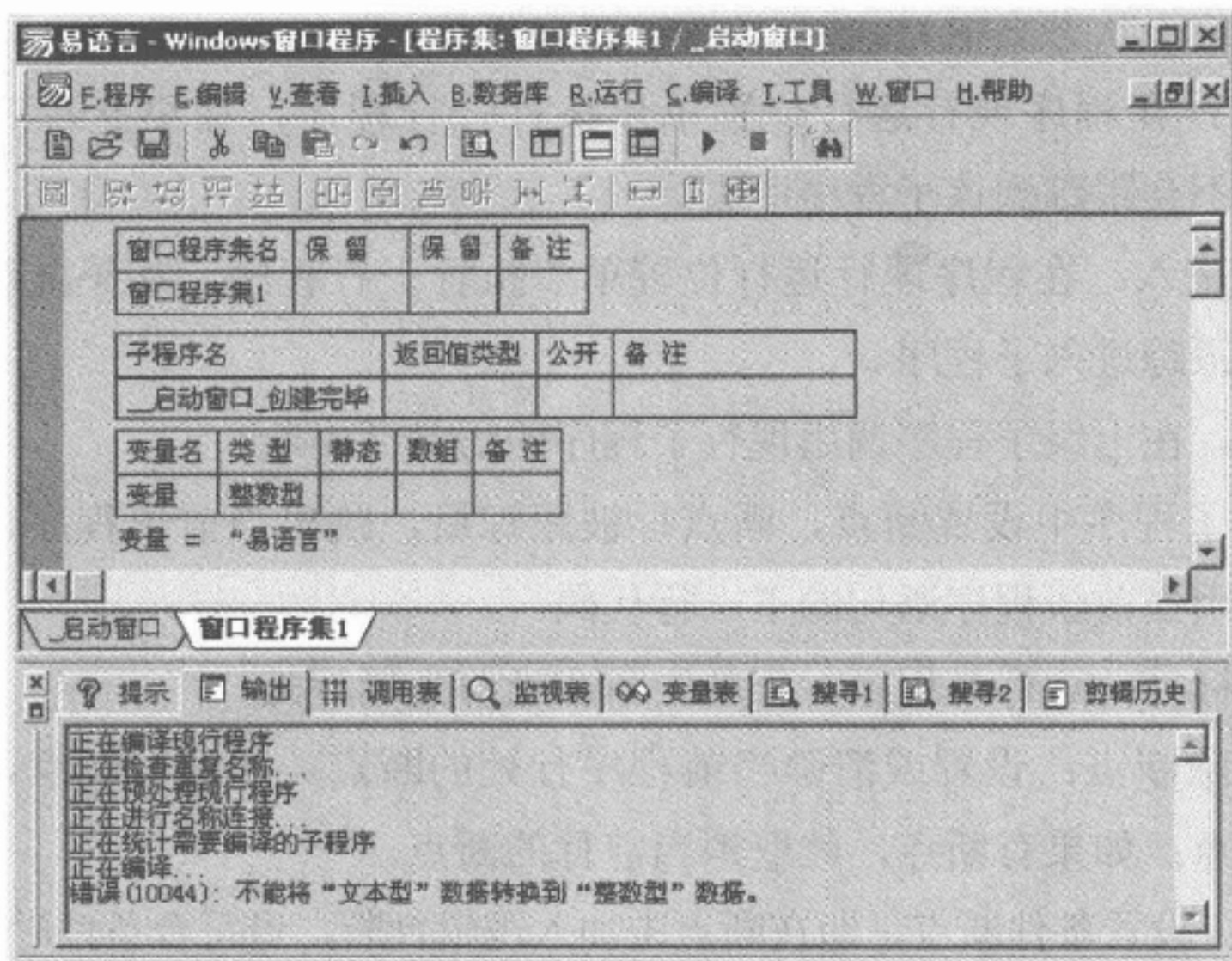


图14-1 初步编译调试

虽然易语言系统提供了预编译调试功能，但也只能指出输入上的格式错误，如数据类型不匹配、指定的变量或组件未找到等。而对于那些编程逻辑上的错误，易语言系统就不会发现，需要借用其他调试方法和调试命令，快速准确地找到错误代码。

## 14.2.2 运行中的调试

易语言提供的运行调试可以在易语言“菜单”→“运行”中查看，如图14-2所示。

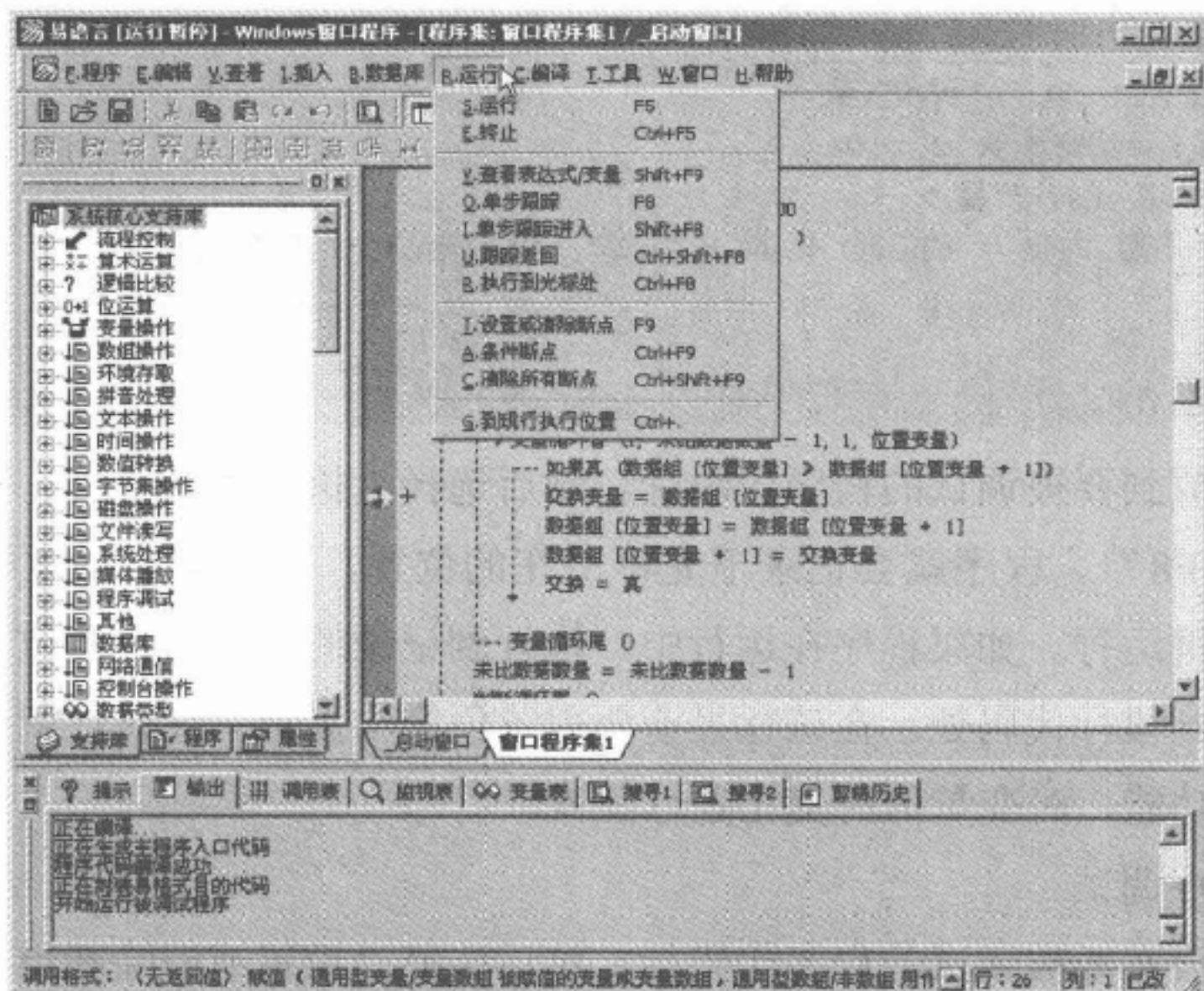


图14-2 易语言运行调试菜单





易语言运行调试菜单说明:

- 查看表达式/变量: 在调试过程中, 查看指定变量的值, 并可以加入到监视表中 (监视表位于状态夹中)。
- 单步跟踪: 在程序现行运行位置单步执行一行程序, 如果此程序行调用了子程序, 系统不会跟踪到该子程序中去。
- 单步跟踪进入: 在程序现行运行位置单步执行一行程序, 如果此程序行调用了子程序, 则跟踪进入子程序。
- 跟踪返回: 在上级子程序调用现行子程序的语句后中断。

在被调用子程序中设置断点, 断点行被跟踪后, 跳出当前子程序, 系统跟踪至上级程序调用当前子程序语句的下一行代码。

- 执行到光标处: 运行易程序, 程序中断在当前光标行处。
- 设置或清除断点: 设置或清除当前程序行处的断点。如没有断点, 就为当前行设置一个断点; 如果有断点, 就取消当前行的断点。
- 条件断点: 设置条件断点, 即在断点中加入逻辑判断, 当符合条件时, 断点才生效。
- 清除所有断点: 清除掉程序中的所有断点。
- 到现行执行位置: 跳到现行即将被执行语句的位置。

下面介绍各种调试功能在程序中的应用。

### 第一步: 执行到光标处

先介绍一种不需要设置断点就可以调试程序的方法——执行到光标处。代码如下:

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
计次变量	整数型			

---> 变量循环首 (1, 100, 1, 计次变量)

--- 如果真 (计次变量 % 2 = 0)

--- 如果真 (计次变量 % 7 = 0)

编辑框1.内容 = 编辑框1.内容 + 到文本 (计次变量) + “,”

--- 变量循环尾 ()

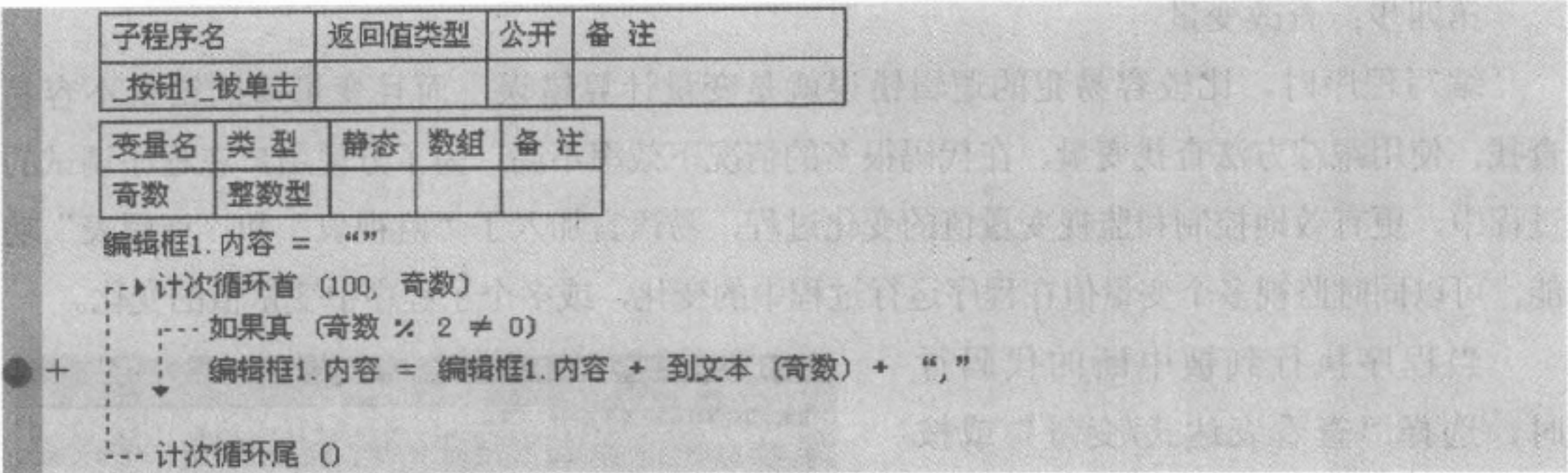
首先将光标放到要被调试的程序代码行。然后选择“执行到光标处”菜单项目, 或按下快捷键“Ctrl+F8”。程序就会记录下光标所在的位置, 并且会自动运行程序, 等同于自动按下“F5”键运行。如果程序在运行中, 执行到记录的光标所在的位置, 程序就会中断, 并将光标移到该行代码处, 在该行最前面用黄色箭头提示。这时, 就可以使用跟踪调试, 查改变量等功能, 对变量值的变化进行监视。

### 第二步: 断点调试

首先将光标放在要被调试的程序代码行。然后选择“设置或清除断点”菜单项目, 或按下快捷键“F9”, 在光标所在行最前面用红色实心圆表示。代码如下:







除了“F9”设置断点外，还可以设置条件断点。可以在断点处进行逻辑判断，只有符合了条件断点设置的条件后，断点才会生效，这无疑提高了程序调试的效率，如图14-3所示。

条件表达式是可以使用“且”或“或”来连接，另外，条件表达式还可以包括四则运算表达式和各级运算符。

“条件断点对话框”中还可以设置“中断前跳过次数”，该跳过次数表示，程序会在断点条件成立后跳过指定次数的断点，然后生效。例如，一个条件断点共有5次符合中断的条件，当设置了中断前跳过次数为2时，则程序会在该断点第3次符合中断条件时，使程序中断。

再按下快捷键“F9”，可以将原有的断点提示清除掉。只要代码执行到标有断点标志的代码行时，程序就会中断。这时，就可以使用跟踪调试、查改变量等方法对变量值的变化进行监视。

### 第三步：跟踪调试

当程序中断后，按下“F8”键，光标及黄色指向箭头会向下一句代码行移动，执行下一句代码。接着不断按“F8”键就可以使程序一步一步执行，直到所在子程序的所有代码被执行完毕。如果需要跟踪到调用的子程序里时，可以使用“单步跟踪进入”；从子程序里跳出，可以使用“跟踪返回”。代码如下：

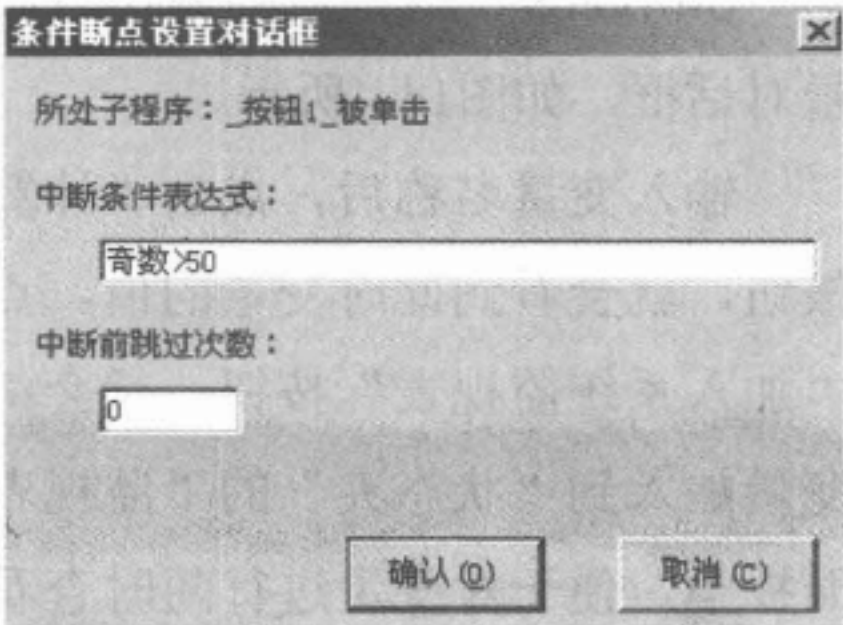
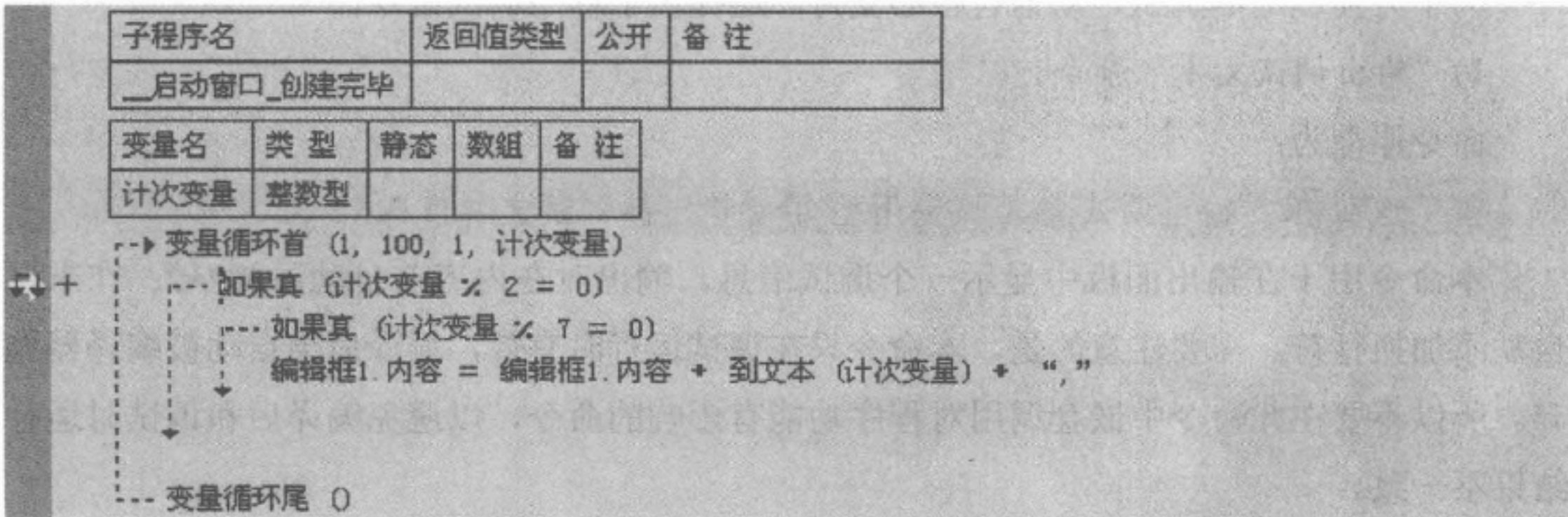


图14-3 设置条件断点





#### 第四步：查改变量

编写程序时，比较容易犯的逻辑错误就是变量计算错误，而且变量值的错误不容易查找，使用跟踪方法查找变量，在代码很多的情况下效率不高。为了方便用户在程序调试的过程中，更有效地控制和监视变量值的变化过程，易语言加入了“监视表”和“调用表”功能，可以同时监视多个变量值在程序运行过程中的变化，或多个子程序中变量值的变化。

当程序执行到被中断的代码行时，选择“查看表达式/变量”或按下“Shift+F9”键，就会弹出查改变量对话框，如图14-4所示。

输入变量名称后，点击“计算”按钮，就会看到此时变量的值。点击“加入系统监视表”按钮，就会将该变量加入到“状态夹”的“监视表”面板中，便于对变量进行随时查看，如图14-5所示。

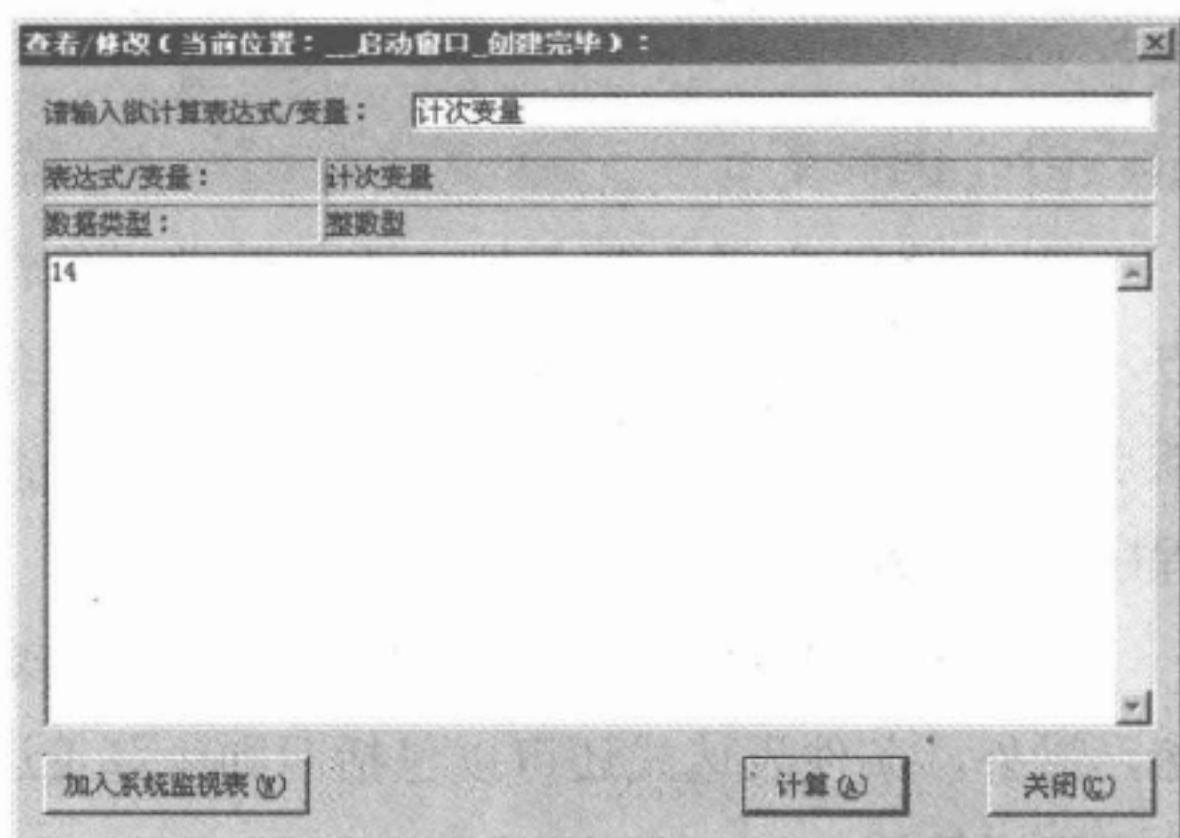


图14-4 查改变量对话框

将状态栏切换到“调用表”面板，如图14-6所示。更改当前监视的子程序可以通过在“调用表”面板中所列出的子程序表中双击选择。

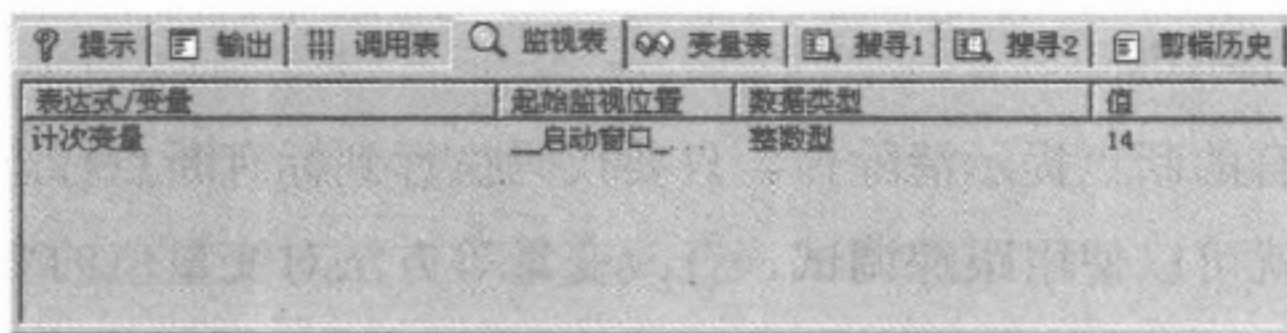


图14-5 监视变量值的变化

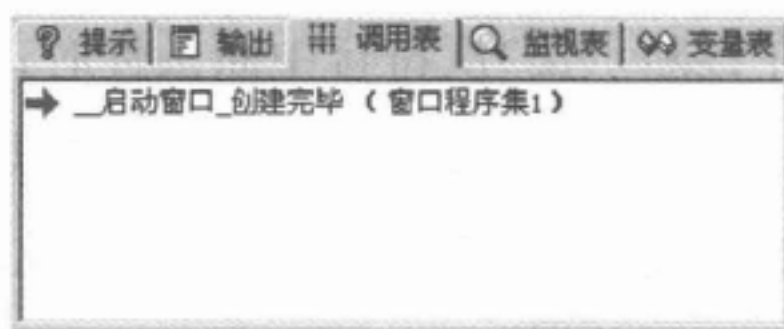


图14-6 调用表

## 14.3 调试命令

为了方便程序的调试，易语言核心支持库还提供了程序调试命令。

### 1) “输出调试文本”命令

命令原型为：

<无返回值> 输出调试文本（通用型 准备输出的调试文本信息，...）

本命令用于在输出面板中显示一个调试信息。输出时在内容前自动添加\*号；在末尾自动添加换行符。需要注意的是，本命令只在调试运行时有效，编译后会自动被编译器抛弃。所以不要在此命令中嵌套调用对程序功能有影响的命令，以避免编译后和调试时运行结果不一致。



参数<1>指定了欲显示的内容。内容的类型可以是文本、日期、数值、逻辑型。其他类型将显示空文本并自动换行。

例如：

输出调试文本（“易语言”）

程序执行后会在输出面板中显示“\* 易语言”，如图14-7所示。

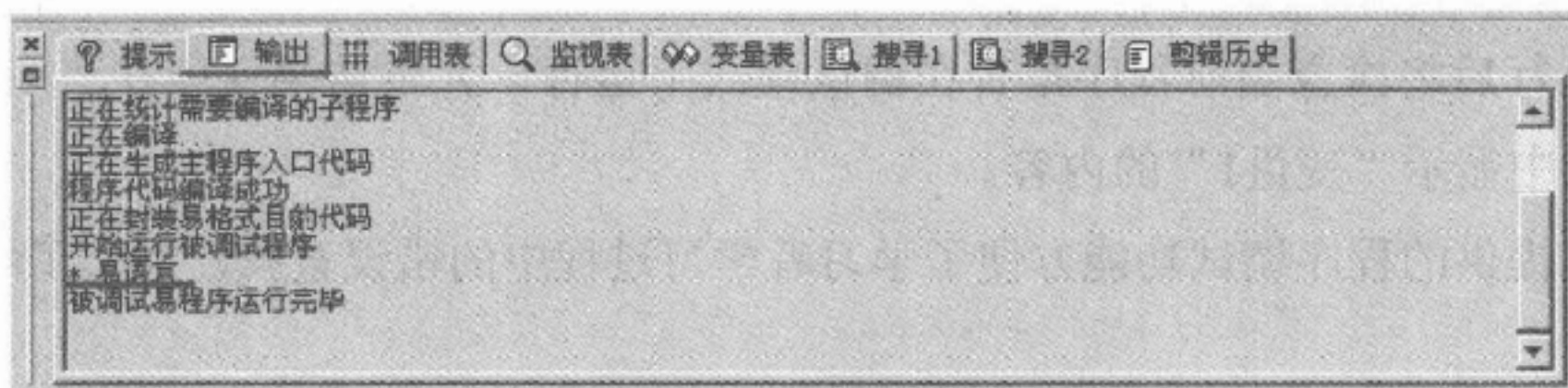


图14-7 输出面板

## 2) “暂停”命令

命令原型为：

<无返回值> 暂停 ()

本命令用于暂时终止程序的运行。效果和设置断点相同，可以查看各变量的值。需要注意的是，本命令只在调试运行时有效，编译后会自动被编译器抛弃。

例如：

暂停 ()

程序运行后执行到暂停命令时中断，进入调试模式。

## 3) “检查”命令

命令原型为：

<无返回值> 检查 (逻辑型 被校验的条件)

本命令用于检查指定条件、逻辑变量或逻辑值。需要注意的是，本命令只在调试运行时有效，编译后会自动被编译器抛弃。所以不要在此命令中嵌套对程序功能有影响的命令，以避免编译后和调试时运行结果不一致。

参数<1>指定了欲检查的内容。若为“假”，则程序中断，进入调试模式；若为“真”，则继续向下执行。

例如：

检查 (1 + 1 = 2)

程序执行后会检查“1+1”是否等于“2”，是则继续执行，否则中断程序进入调试模式。

## 4) “是否为调试版”命令

命令原型为：

<逻辑型> 是否为调试版 ()

本命令用于检查所在程序是否在调试模式下运行。如果当前易程序执行文件为易语言编辑环境调试运行的程序，返回真，否则表明为发布版本，返回假。





例如：

变量名	类型	静态	数组	备注
变量1	逻辑型			

变量1 = 是否为调试版 0

» 输出调试文本 (变量1)

程序执行后将检查当前程序是否在调试模式下运行，并将结果保存在“变量1”中；在输出面板中显示“变量1”的内容。

易语言提供的程序调试功能方便了学习者学习过程中的错误查找，加快了编程者开发软件的速度。

## 14.4 小结

每一个程序一般都要经过反复的调试和修改才能最终完成。语法错误一般都能很快发现，逻辑错误需要仔细的查找。使用调试方法可以快速找出程序问题所在。易语言提供了预编译调试功能，可以检查现行代码的输入错误；设置断点功能可以查改变量；跟踪调试功能不但可以逐行代码调试运行，还可以进入到被调用的子程序中。程序调试类命令只在调试运行时有效，编译时将被自动忽略。

## 14.5 习题

- 14-1 程序调试是\_\_\_\_\_。
- 14-2 程序中出现的错误一般分为\_\_\_\_\_和\_\_\_\_\_。
- 14-3 断点调试的快捷键是\_\_\_\_\_。
- 14-4 单步跟踪调试的快捷键是\_\_\_\_\_。
- 14-5 \_\_\_\_\_命令用于在输出面板中显示一个调试信息。输出时在内容前自动添加\_\_\_\_\_号；在末尾自动添加\_\_\_\_\_。
- 14-6 \_\_\_\_\_命令用于暂时终止程序的运行。效果和设置断点相同，可以查看各变量的值。需要注意的是，本命令只在\_\_\_\_\_运行时有效，编译后会自动被编译器抛弃。
- 14-7 \_\_\_\_\_命令用于检查所在程序是否在调试模式下运行。
- 14-8 易语言的调试方法有哪些？







## 第十五章 程序的编译与发布

### 本章目标

在本章结束时，我们能够：

- 了解易语言4.X版的编译方式
- 了解易语言5.X版的编译方式
- 了解易语言中编译方式的异同之处
- 掌握配置要编译的易语言程序
- 掌握制作易语言程序安装包





## 15.1 编译前的配置

当程序编写完毕后，如果想要在其他电脑上安装并运行本程序。就需要对其进行编译。在编译前，需要先配置一下程序的信息。

### 第一步：系统配置

编译程序前可以在系统配置对话框（菜单“工具”→“系统配置”菜单项）中设置编译选项，如图15-1所示。

各选项说明如下：

- 是否启用快速数组访问方式：可以提高数组或字节集的访问速度，但数组越界错误不会被系统处理。
- 调用DLL命令后是否检查堆栈错误：可以避免由于调用一些错误的外部DLL命令而导致堆栈错误。
- 编译时是否检查死循环代码：可以检查死循环代码。
- 是否使用Windows通用组件库6.0版：可以使用Windows系统自带的通用组件库6.0版。
- 严格的重复名称检查：可以严格的检查代码中重复的名称。
- 非独立编译后是否询问写出相关库：可以在易代码非独立编译后，弹出提示框，询问是否将与编译程序相关的支持库写到编译程序所在的文件夹。

易语言还充分考虑了编译程序的安全性。为了增大程序被破解的难度，加强程序的保密性，保护目的程序的安全，加入了插入花指令和代码随机打乱的功能，如图15-2所示。

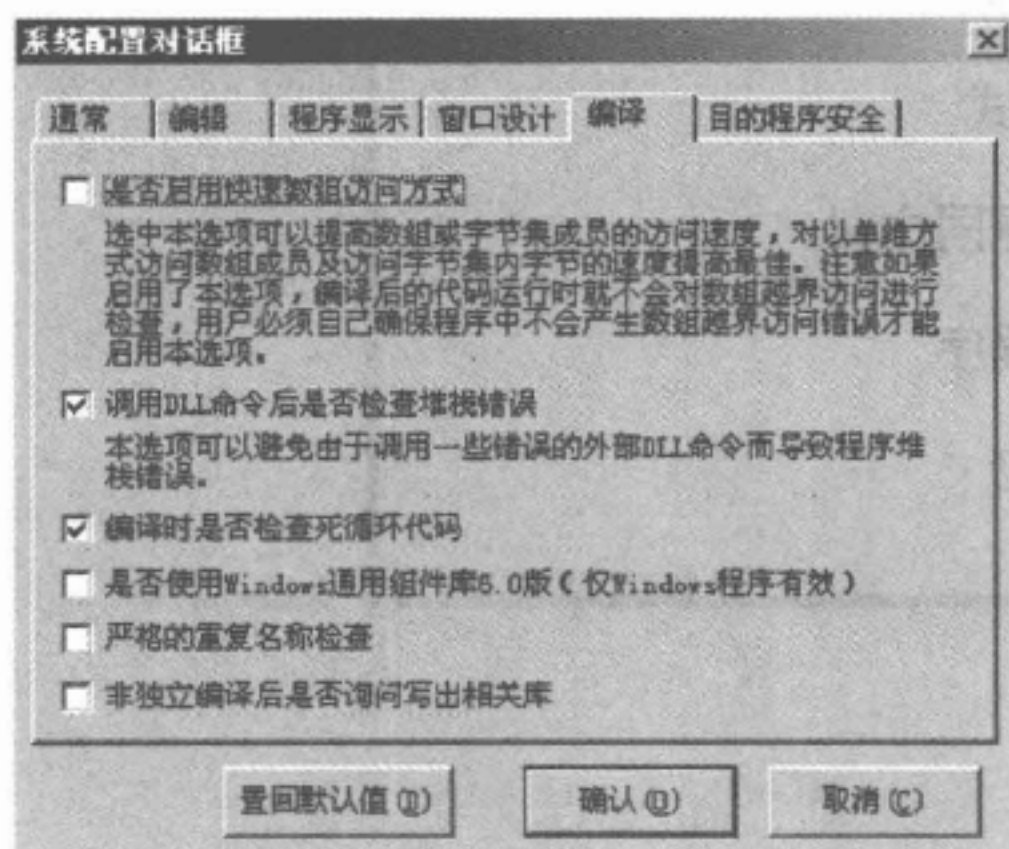


图15-1 系统配置对话框编译项

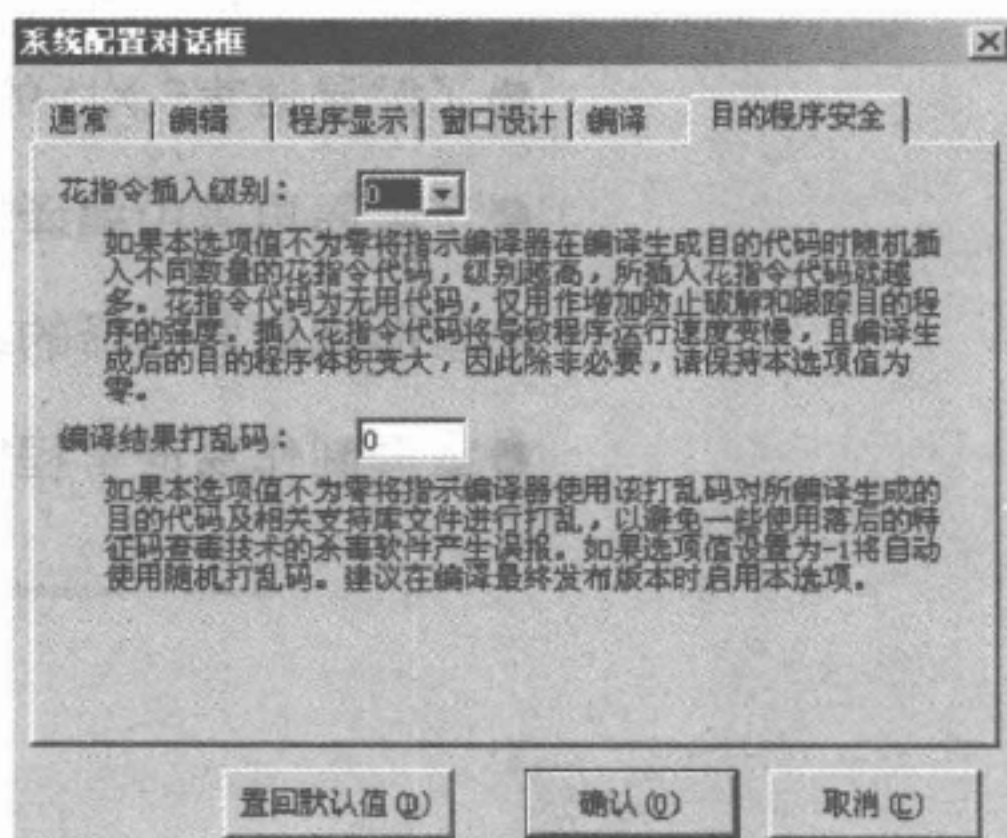


图15-2 系统配置对话框目的程序安全项





## 第二步：程序配置

还可以在程序配置对话框（菜单“程序”→“配置”菜单项）中设置程序版本信息、作者信息、源程序密码等。

各选项说明如下：

- 通常项：设置关于程序的一些信息。编译程序的图标也可以在此设置，如图15-3所示。
- 作者信息项：设置程序作者的相关信息，如图15-4所示。
- 其它项：在此可以设置程序源代码的密码，用以保护源代码，如图15-5所示。

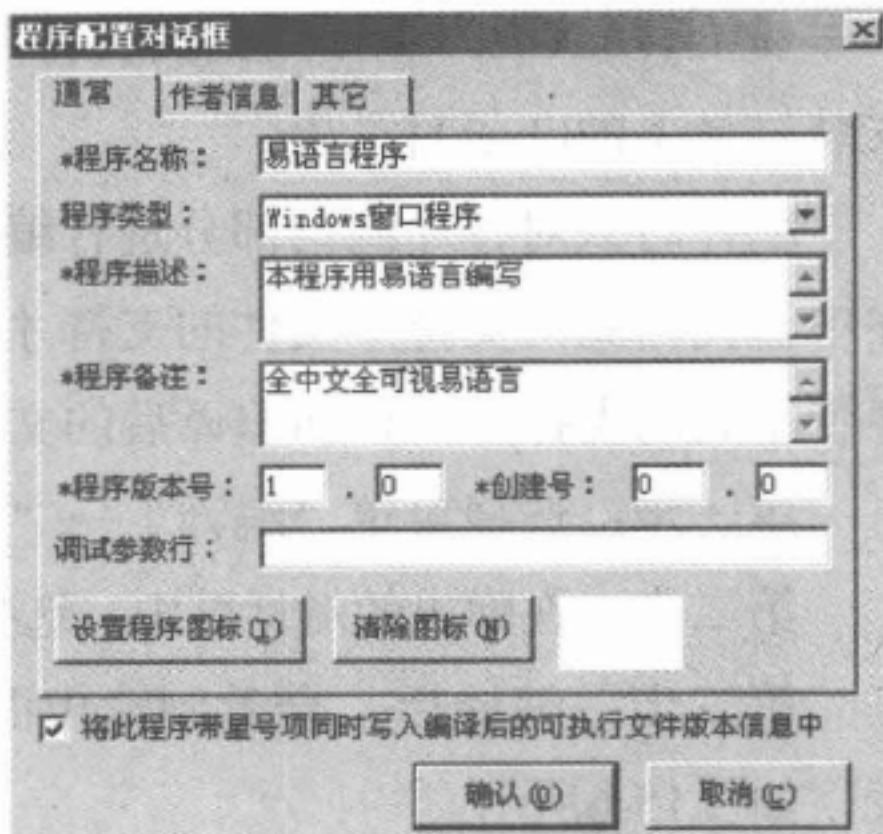


图15-3 程序配置对话框通常项

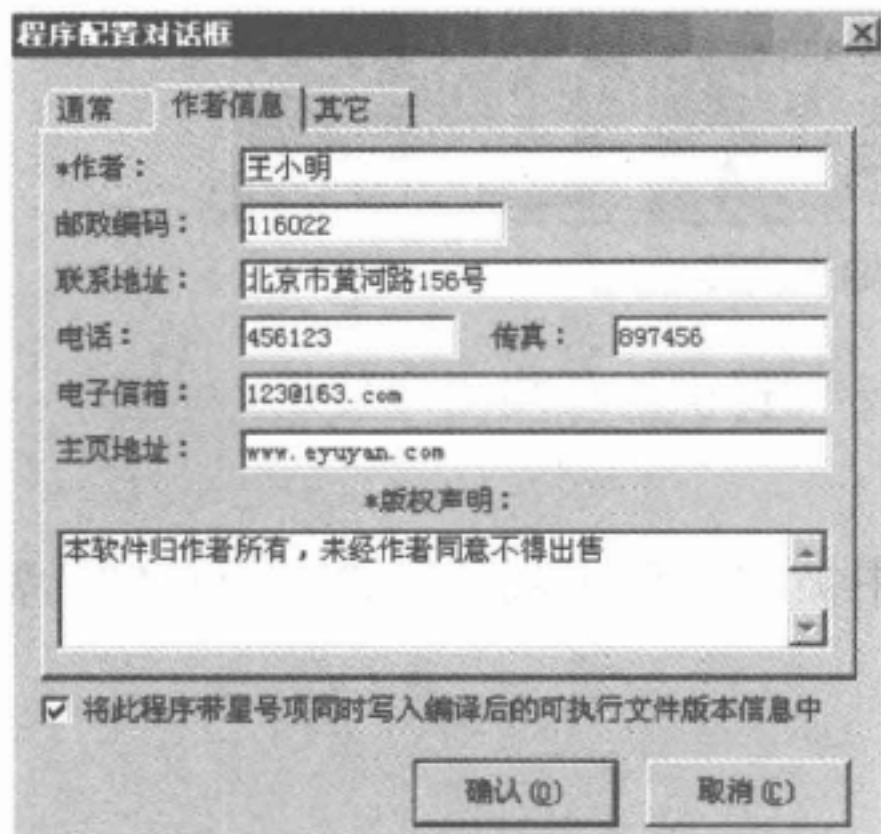


图15-4 程序配置对话框作者信息项

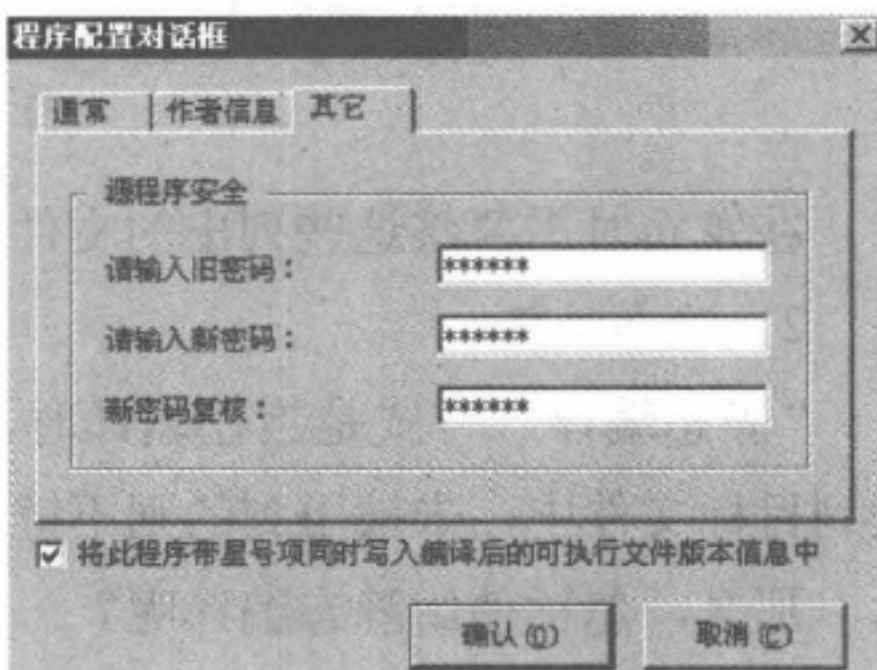


图15-5 程序配置对话框其他项

配置完程序的信息，就可以编译程序了。

## 15.2 易语言程序的编译

2010年2月1日，易语言公司发布了易语言5.0正式版本，对易语言的编译方式进行了非常大的改动，实现了易语言的静态编译。静态编译后的易语言可执行程序（exe）和动态链接库（DLL）运行时不再依赖任何支持库文件，同时相对4.X版本下的独立编译后的文件尺寸更小；PE结构更合理（取消了“易格式体”）；加载速度更快。

**注解：** 只有拥有易语言正版授权才能编译易语言程序。

### 15.2.1 易语言5.X版本的编译

易语言5.X版本编译方式有两种：编译和静态编译。



### 1) 编译

易语言5.X版本编译用到的支持库是动态支持库。动态支持库存放在易语言5.X版本的安装目录下的lib子目录中。

编译时只对程序源码部分进行编译而不包含易语言的支持库。编译后的程序必须有程序中所使用到的支持库文件的支持才可以运行。所以编译后的程序如果要发布，必须将程序使用到的支持库文件和编译后的文件一同发布。

第一步：选择菜单“编译”→“编译”选项，或者按下“F7”键。

第二步：在弹出的对话框中，选择所要保存的文件路径并输入目标文件名，保存文件。

第三步：编译后，会看到输出面板中有编译成功的信息，以及运行时依赖的文件列表，如图15-6所示。

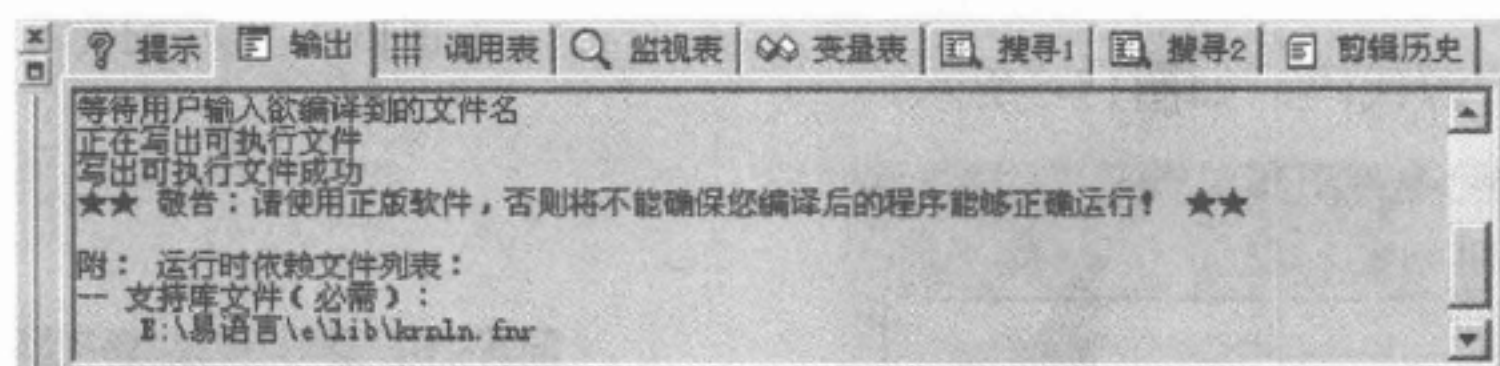


图15-6 编译后的输出信息

在发布时需要将这些列出的文件与目标文件一同提供给最终用户使用。

### 2) 静态编译

“静态编译”方式是指在编译时，把支持库中被程序使用到的有效代码按实际需要链接到目标文件中，未被用到的则不链接。

那么，怎样使用静态编译呢？

易语言5.X版本提供了静态编译的配置文件toos\link.ini，并对易语言5.X版本中静态编译的配置做了详细的说明。在静态编译前，必须指定静态编译时使用的外部链接器，并给出完整路径。

第一步：将link.ini配置文件中的{;linker="C:\full\path\link.exe"}文本行行首的半角分号(;)去掉。

第二步：将外部链接器的路径修改成指定的外部链接器的完整路径。例如linker="C:\link.exe"。外部链接器有vc6、gcc4、elink、alink等多种类型，可以到网上下载使用。

易语言5.X版本的静态编译用到的支持库是静态支持库。静态支持库存放在易语言5.X版本的安装目录的static\_lib目录里。

## 15.2.2 易语言4.X版本的编译

易语言4.X版本的编译方式有两种：编译和独立编译。

易语言4.X版本的编译方式和易语言5.X版本的编译方式是一样的。

易语言4.X版本易语言的编译和独立编译用到的支持库是动态支持库。动态支持库存放在易语言4.X版本的安装目录的lib目录里。



易语言的“独立编译”可把易程序独立运行所需要的文件全部封装到可执行文件中，使其可以完全独立的运行。程序中所用到的支持库会自动全部封装进去，而其他外部文件，如DLL、OCX等，可以在编译时，由开发人员自由选择是否封装到可执行文件之中。具体方法如下：

第一步：选择菜单“编译”→“独立编译”选项。

第二步：在弹出的窗口中显示了将要保存到EXE文件中的所有文件，可以选择一项或多项进行编译，而灰色项目为必需的。没有选择编译的文件，要随同EXE文件一起提供给使用者，如图15-7所示。

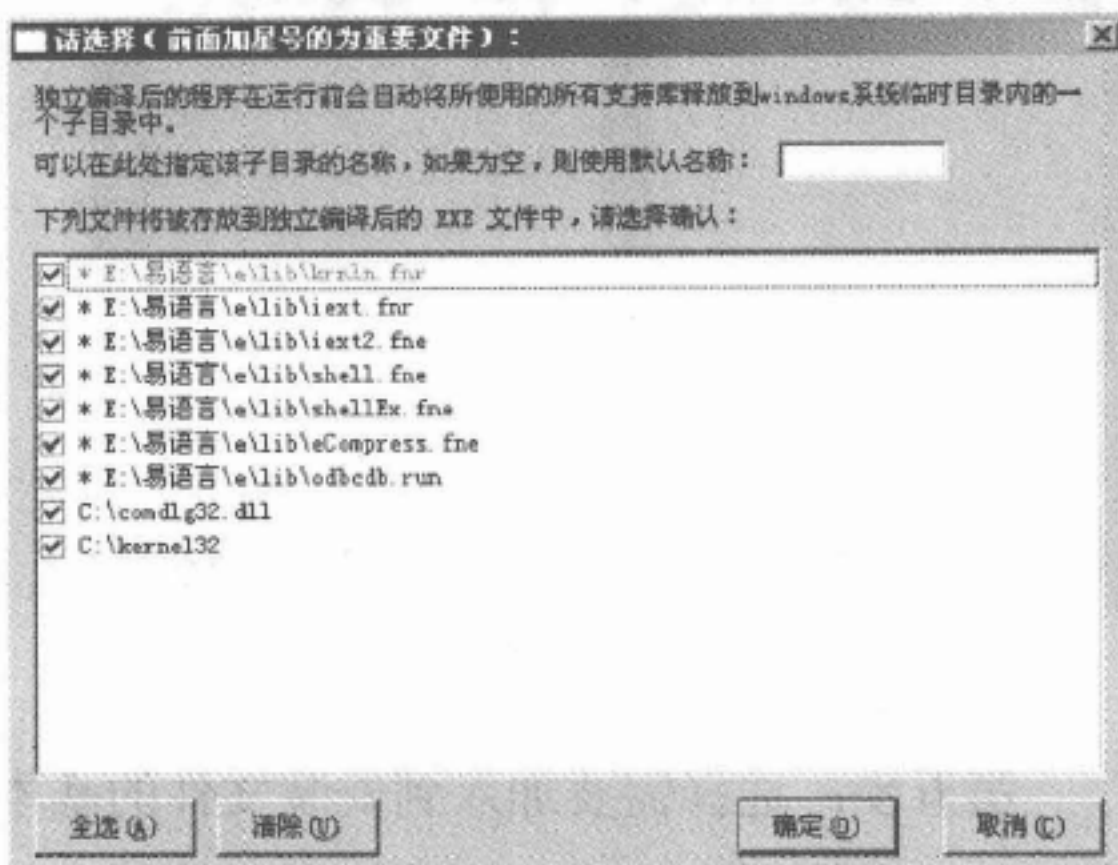


图15-7 选择独立编译项目

第三步：在弹出的对话框中，选择所要保存文件的路径，然后保存为EXE文件。如果没有在“程序配置对话框”中设置图标，编译后的可执行文件将使用易语言默认图标样式。编译后，会看到输出面板中有编译成功的信息，如图15-8所示。

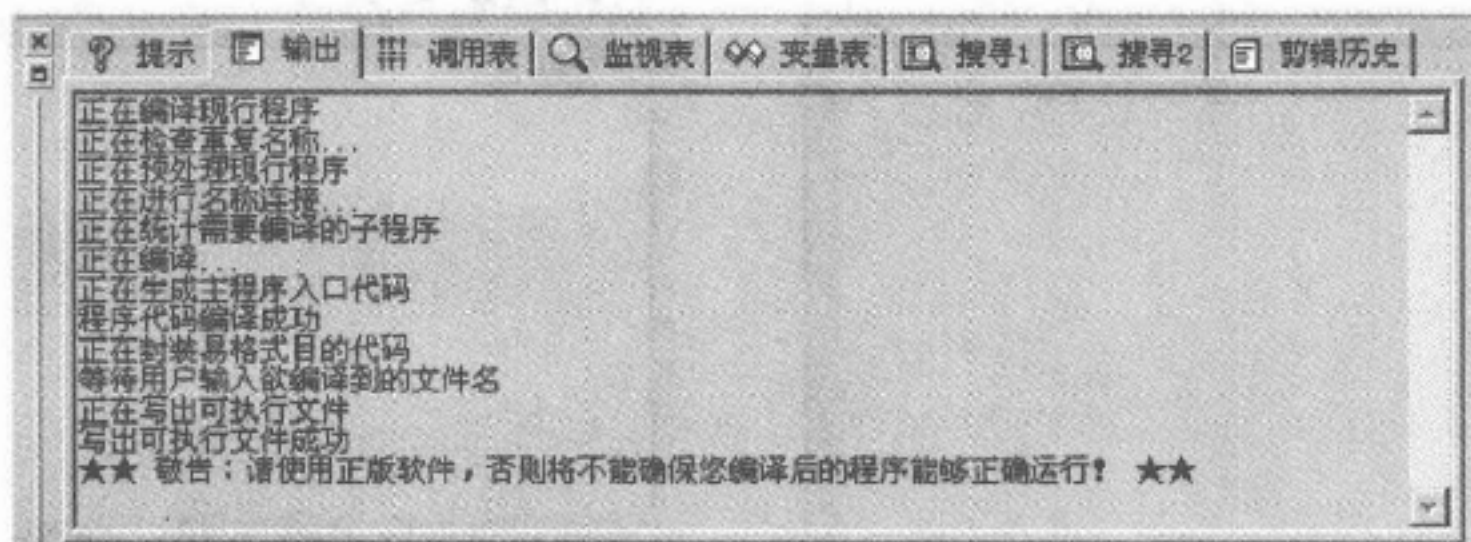


图15-8 独立编译成功输出的信息

## 15.3 编译生成安装软件

为方便程序的发布，易语言提供了一个标准的安装软件制作向导程序，通过友好的向导界面提示用户安装易程序，同时也避免了在程序发布时漏掉文件。程序制作成安装软件前要保存，如果程序没有保存，系统会自动弹出一个信息框提示保存。详细方法如下。

第一步：选择菜单“编译”→“编译生成安装软件”选项。

**第二步：**确定即将生成的安装软件的保存路径和文件名称，点击“保存”按钮继续。

第三步：添加安装软件时的窗口标题以及软件作者的声明和许可协议，点击“下一步”按钮继续，如图15-9所示。

第四步：设置安装窗口的外观，设置完可以点击“预览”按钮预先查看窗口外观的效果。点击“下一步”继续，如图15-10所示。



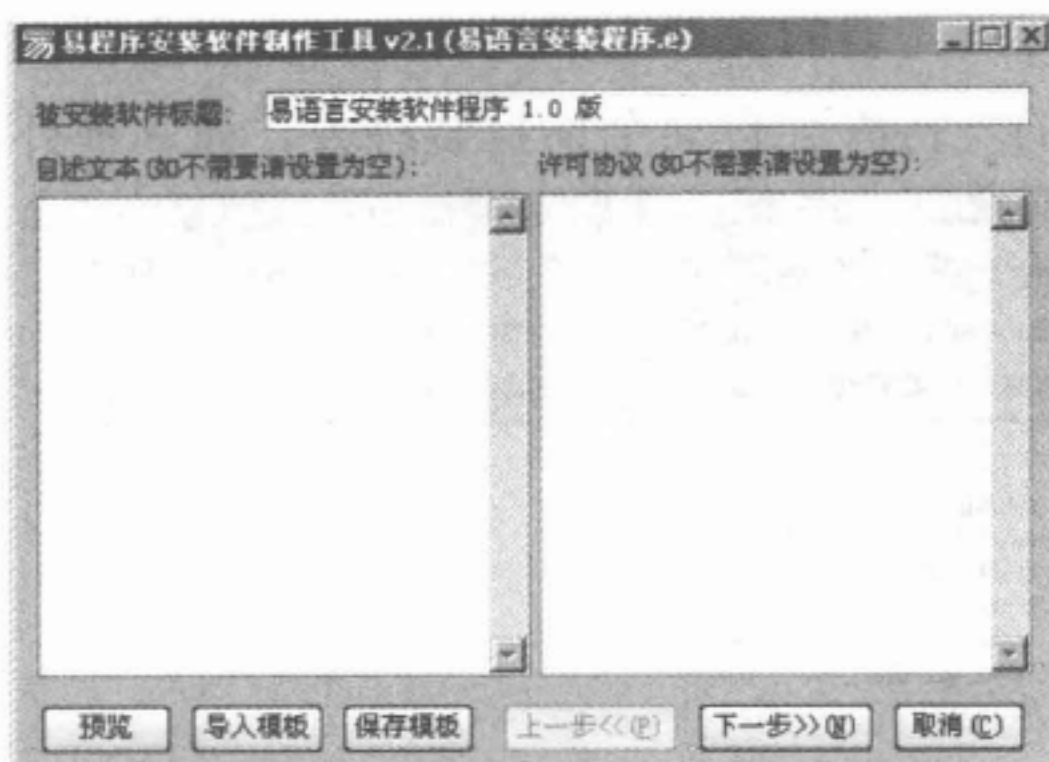


图15-9 添加作者声明和许可协议

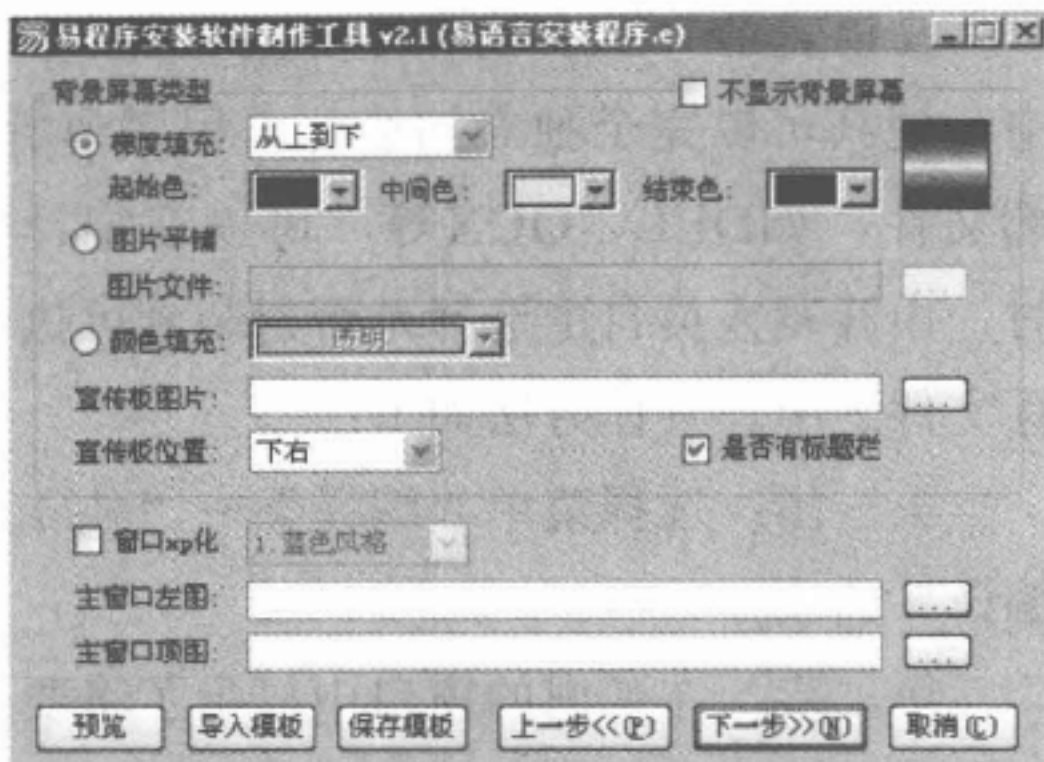


图15-10 设置安装软件的窗口外观

第五步：选择需要加入到安装文件的基本文件，灰色为必选项，黑色选项如果不选中加入，则发布软件时该文件要另外提供给用户。点击“下一步”继续，如图15-11所示。

第六步：添加其他相关文件至不同的安装目录中，点击“下一步”继续，如图15-12所示。



图15-11 选择安装软件的内部文件

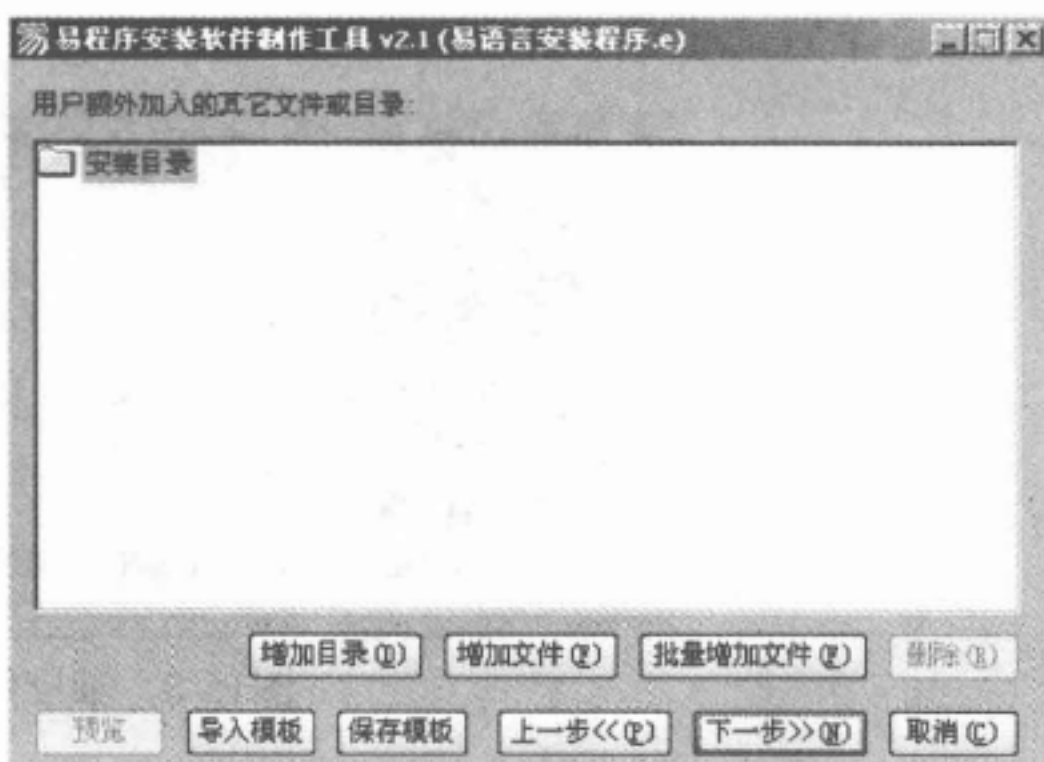


图15-12 添加安装软件的外部文件

第七步：设置软件的启动及卸载方式，点击“下一步”继续，如图15-13所示。

第八步：设置软件默认的安装目录及相关选项，点击“下一步”继续，如图15-14所示。



图15-13 软件的启动及卸载方式

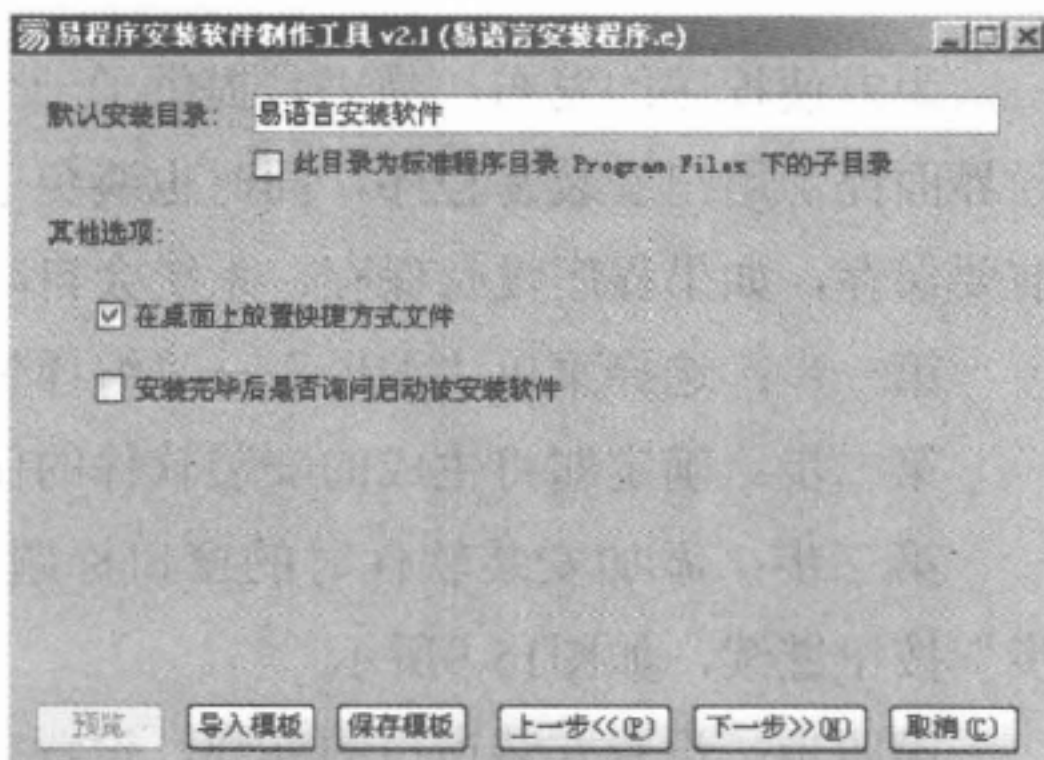


图15-14 设置软件安装时的默认安装目录





第九步：确定安装软件的文件属性。点击“确定”按钮就完成安装软件的制作了，如图15-15所示。

第十步，编译安装完成之后，会弹出一个“生成安装软件成功”的提示窗口，如图15-16所示。

安装向导编译所生成的安装文件使用了默认图标样式，如图15-17所示。

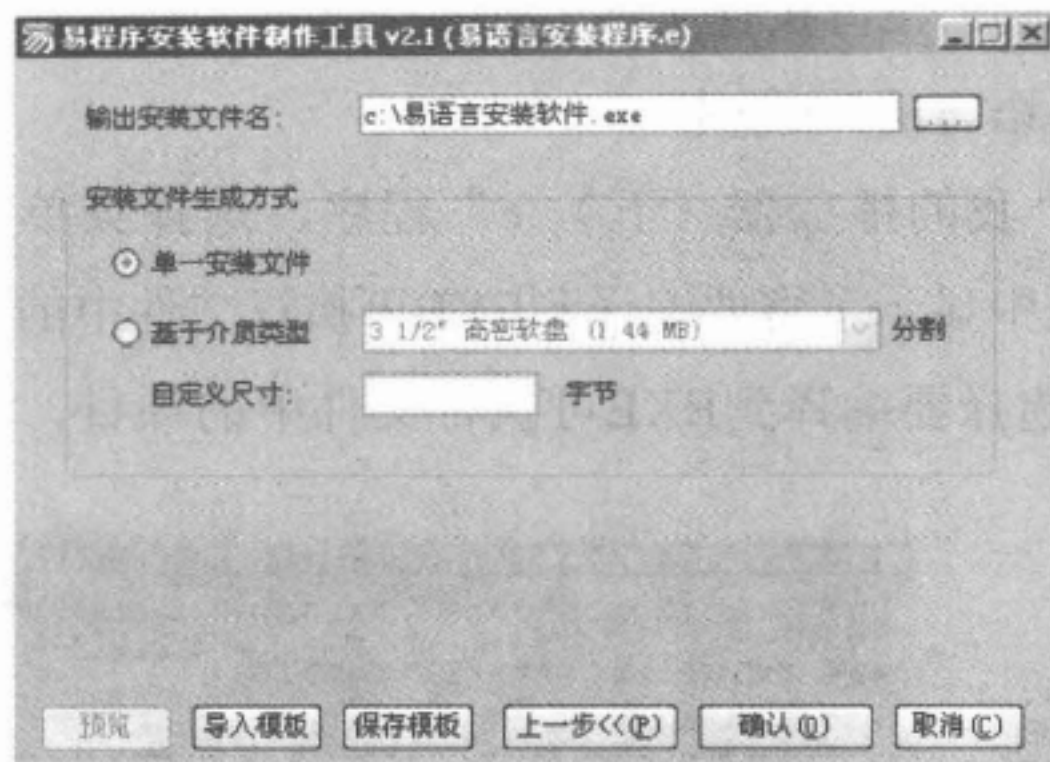


图15-15 选择保存制作后的安装文件的目录

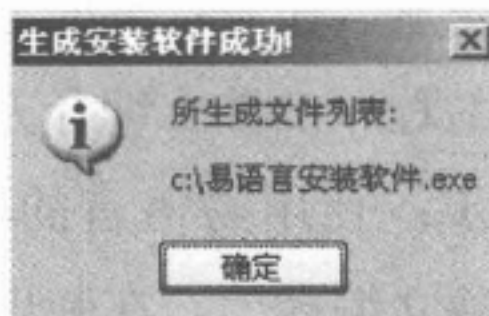


图15-16 提示生成安装软件成功



图15-17 安装软件

**注意：**制作成安装软件的易程序是采用编译方式编译的。

## 15.4 编译安装应用实例

下面通过“我的播放器（五）.e”程序，进行编译安装操作，加深读者对易语言编译方式的理解。

(1) 易程序的编译方式：适用于易语言的4.X版本和5.X版本。

第一步：使用易语言的4.X版本打开“我的播放器（五）.e”程序，选择菜单“编译”→“编译”，在弹出的输入文件名对话框中，选择所要保存的文件路径并输入目标文件名，保存文件，如图15-18所示。

第二步：保存后，会弹出询问提示框，提示是否将源程序所使用到的相关依赖文件（一般为易语言支持库）写到编译后的可执行文件所保存的目录中，如图15-19所示。



图15-18 输入要保存的文件名

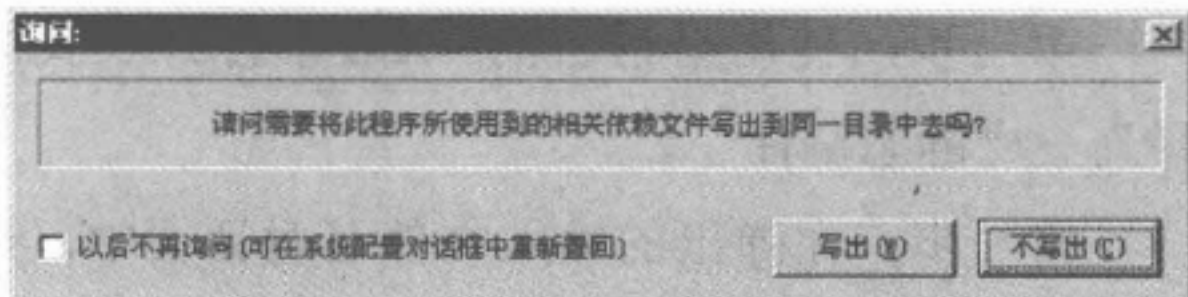


图15-19 询问提示框



第三步：在包含有易语言系统的机器上使用，一般选择“不写出”；在发布给用户使用时，一般选择“写出”，因为用户的机器上不一定包含易语言系统，或包含易语言系统但易语言支持库的版本不一样。

关闭询问提示框，会看到输出面板中有编译成功的信息，以及运行时依赖的文件列表，如图15-20所示。

编译完成，在系统桌面上就会看到保存的EXE可执行文件了。

(2) 易程序的独立编译方式：适用于易语言的4.X版本。

第一步：使用易语言的4.X版本打开“我的播放器（五）.e”程序，选择菜单“编译”→“独立编译”，在弹出的选择对话框中显示了将要编译到EXE可执行文件中的所有易语言支持库文件及其他相关依赖文件，选择要编译到EXE可执行文件中的项目，如图15-21所示。

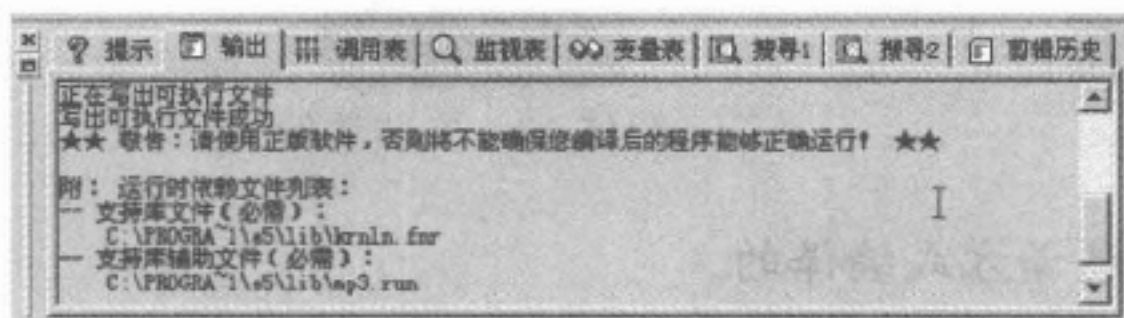


图15-20 编译成功输出的信息

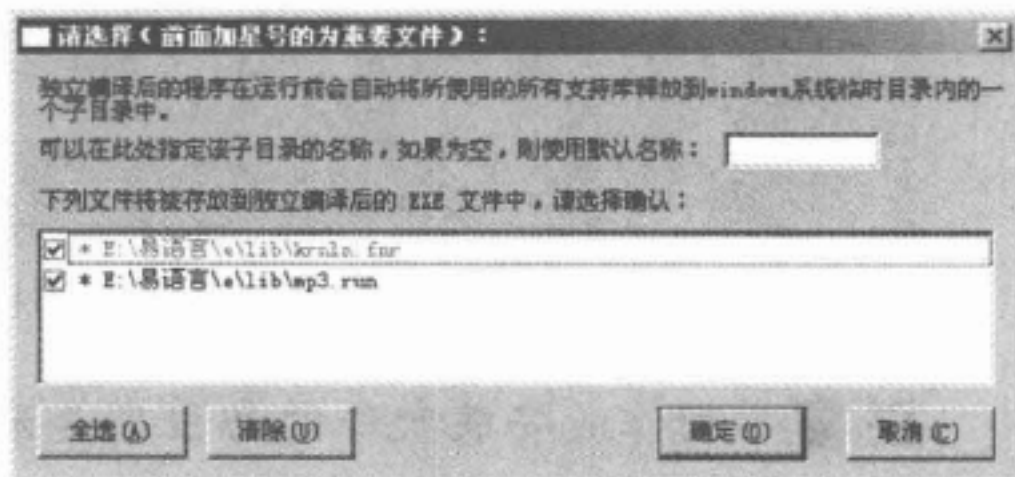


图15-21 选择独立编译项目

第二步：可以选择一项或多项进行编译，而灰色项目为必需的。没有选择编译的文件，要随同EXE可执行文件一起提供给使用者。在弹出的输入文件名对话框中，选择所要保存的文件路径并输入目标文件名，保存文件，如图15-22所示。

文件保存后，会看到输出面板中有编译成功的信息，如图15-23所示。



图15-22 输入要保存的文件名

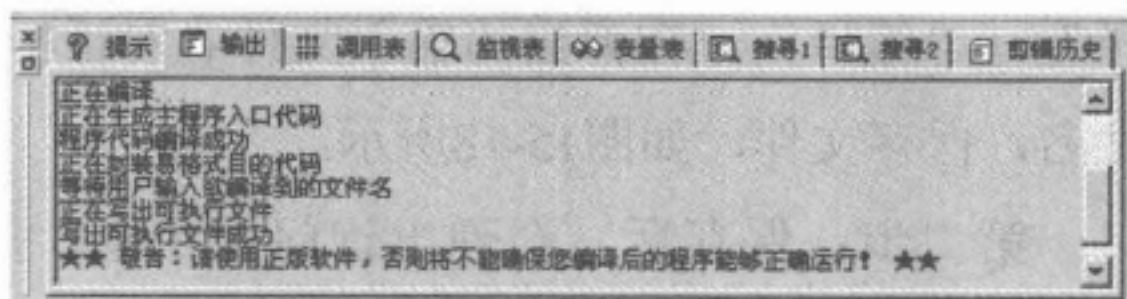


图15-23 独立编译成功输出的信息

编译完成，在系统桌面上就会看到保存的EXE可执行文件了。

(3) 易程序的静态编译方式：适用于易语言的5.X版本。

第一步：使用易语言的5.X版本打开“我的播放器（五）.e”程序，选择菜单“编译”→“静态编译”。

第二步：在弹出的输入文件名对话框中，选择所要保存的文件路径并输入目标文件名，保存文件，如图15-24所示。



文件保存后，会看到输出面板中有编译成功的信息，如图15-25所示。



图15-24 输入要保存的文件名

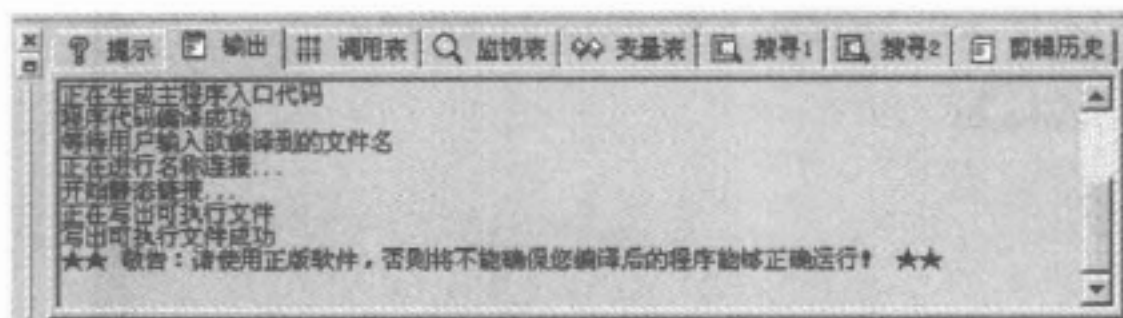


图15-25 静态编译成功输出的信息

编译完成，在系统桌面上就会看到保存的EXE可执行文件了。

(4) 制作易程序的安装包：适用于易语言的4.X版本和5.X版本。

第一步：使用易语言的4.X版本或5.X版本打开“我的播放器（五）.e”程序，选择菜单“编译”→“编译生成安装软件”，在弹出的输入文件名对话框中选择所要保存的文件路径并输入目标文件名，保存文件，如图15-26所示。



图15-26 输入要保存的文件名

第二步：添加安装软件时的窗口标题以及软件作者的声明和许可协议，点击“下一步”按钮继续，如图15-27所示。

第三步：设置安装窗口的外观，设置完可以点击“预览”按钮预先查看窗口外观的效果。点击“下一步”继续，如图15-28所示。

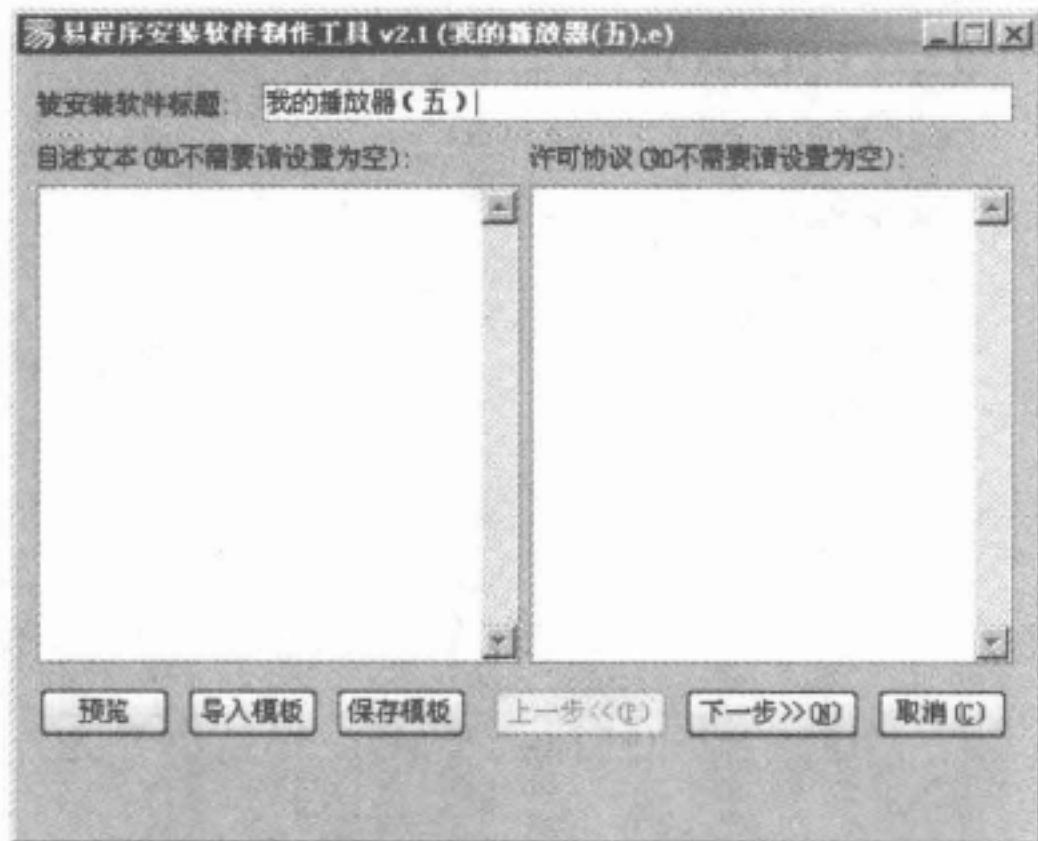


图15-27 添加作者声明和许可协议

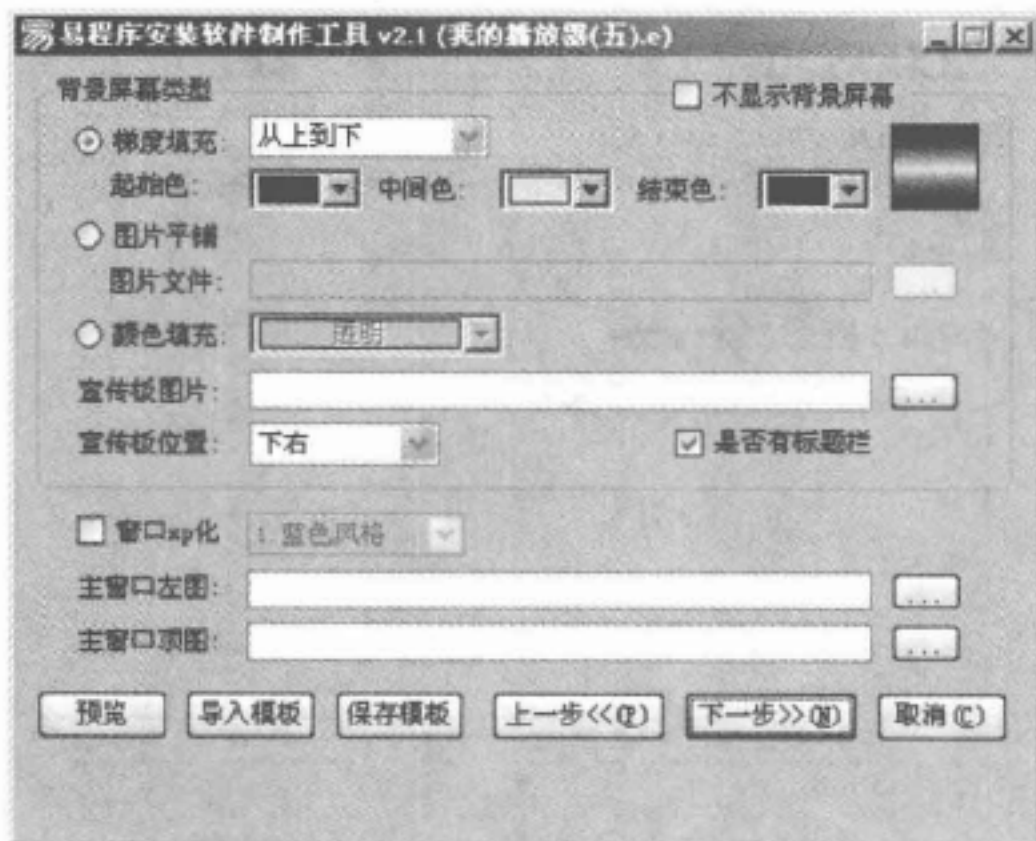


图15-28 设置安装软件的窗口外观

第四步：选择需要加入到安装文件的基本文件，灰色为必选项，黑色选项如果不选中加入，则发布软件时该文件要另外提供给用户。点击“下一步”继续，如图15-29所示。



第五步：添加其他相关文件至不同的安装目录中，点击“下一步”继续，如图15-30所示。

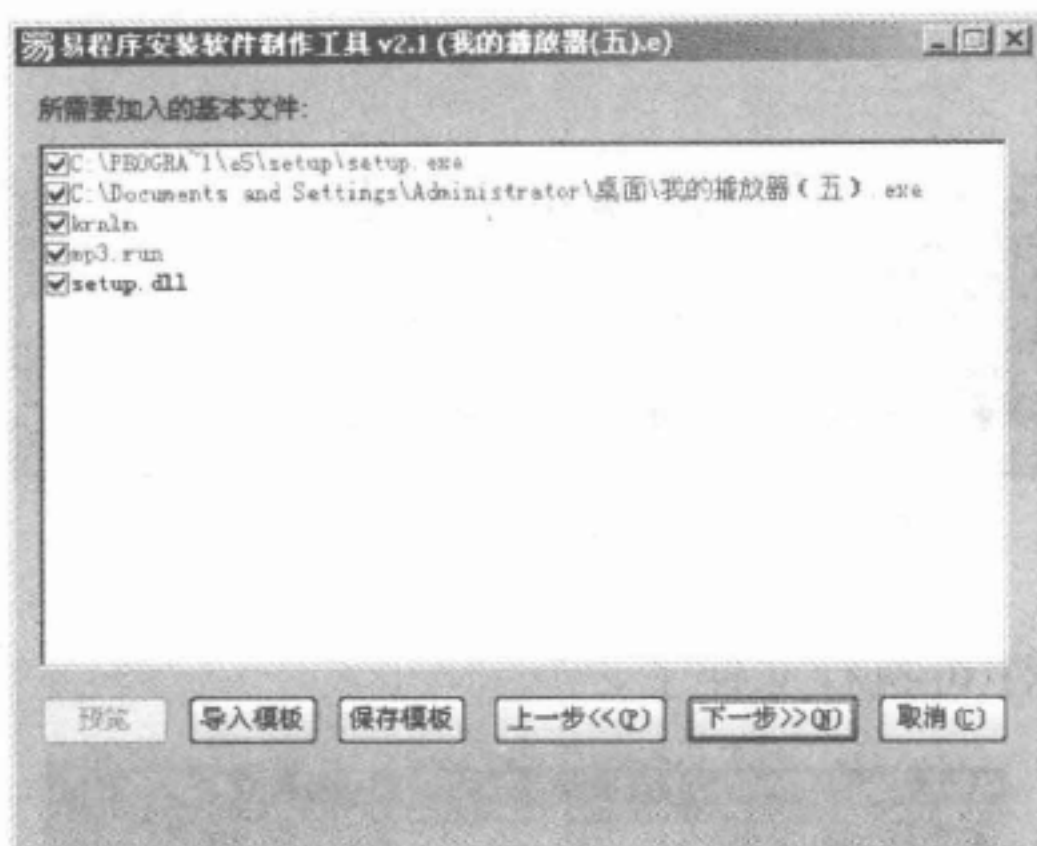


图15-29 选择安装软件的内部文件

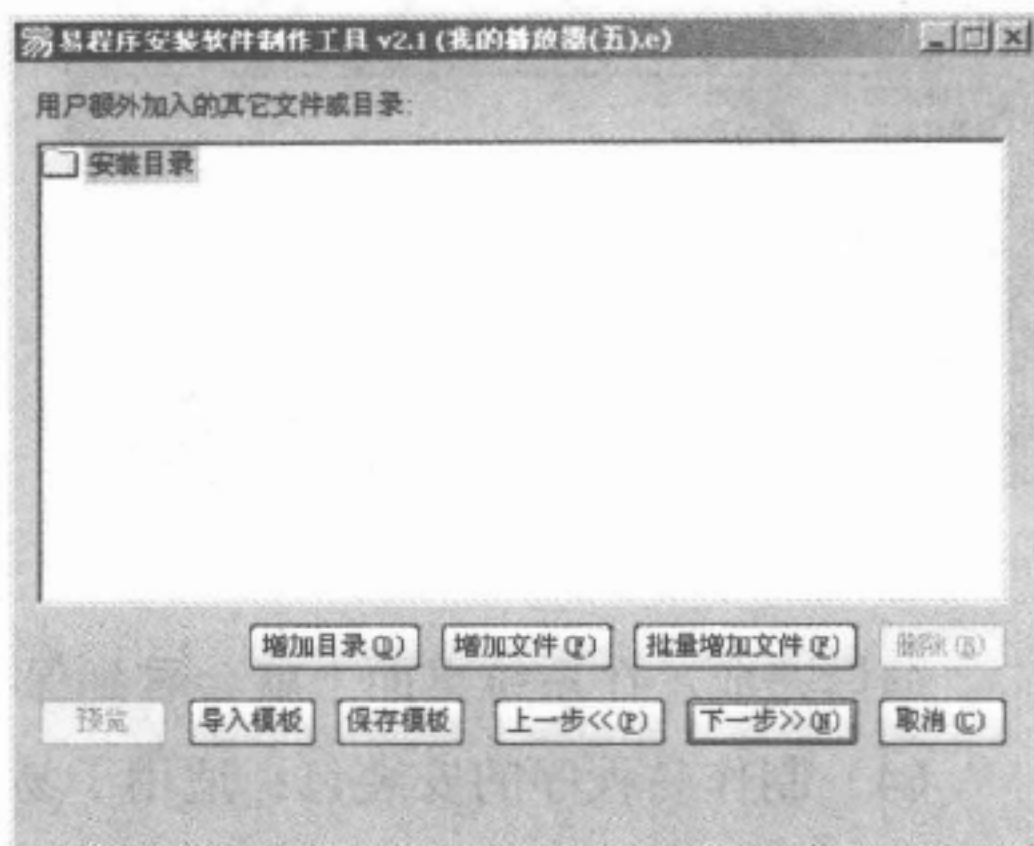


图15-30 添加安装软件的外部文件

第六步：设置软件的启动及卸载方式。点击“下一步”继续，如图15-31所示。

第七步：设置软件默认的安装目录及相关选项，点击“下一步”继续，如图15-32所示。

第八步：确定安装软件的路径及文件名。点击“确定”就完成安装软件的制作了，如图15-33所示。

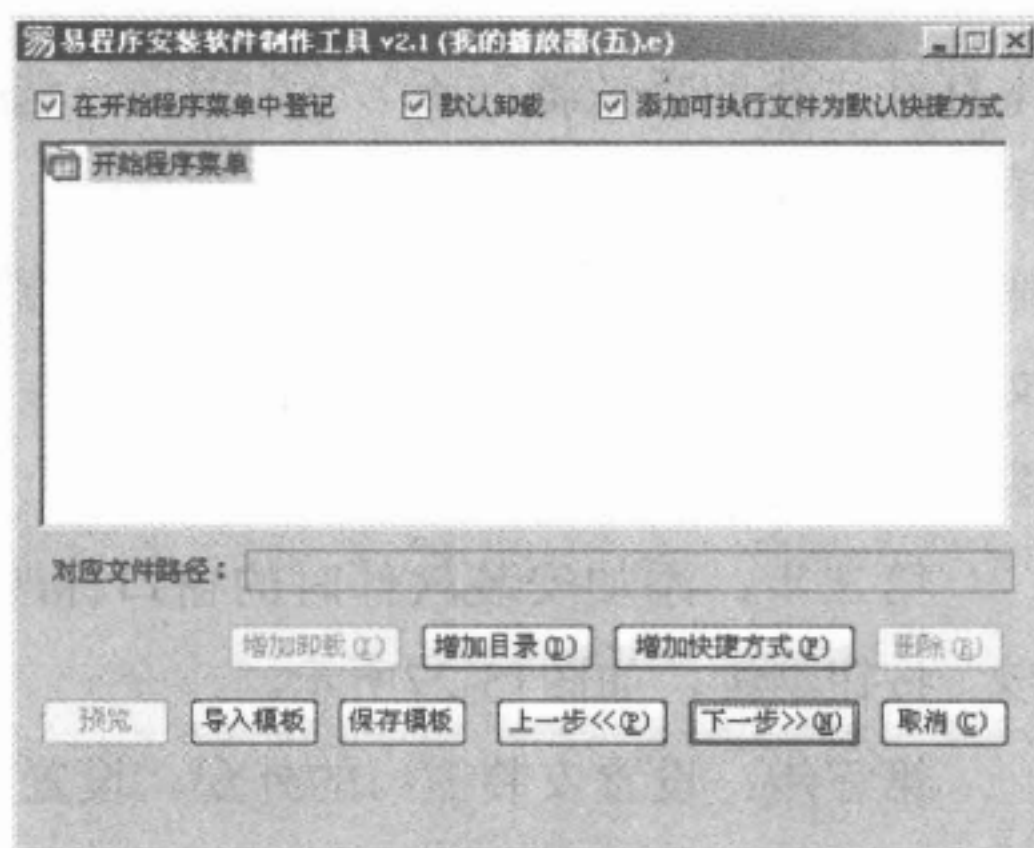


图15-31 设置软件的启动及卸载方式

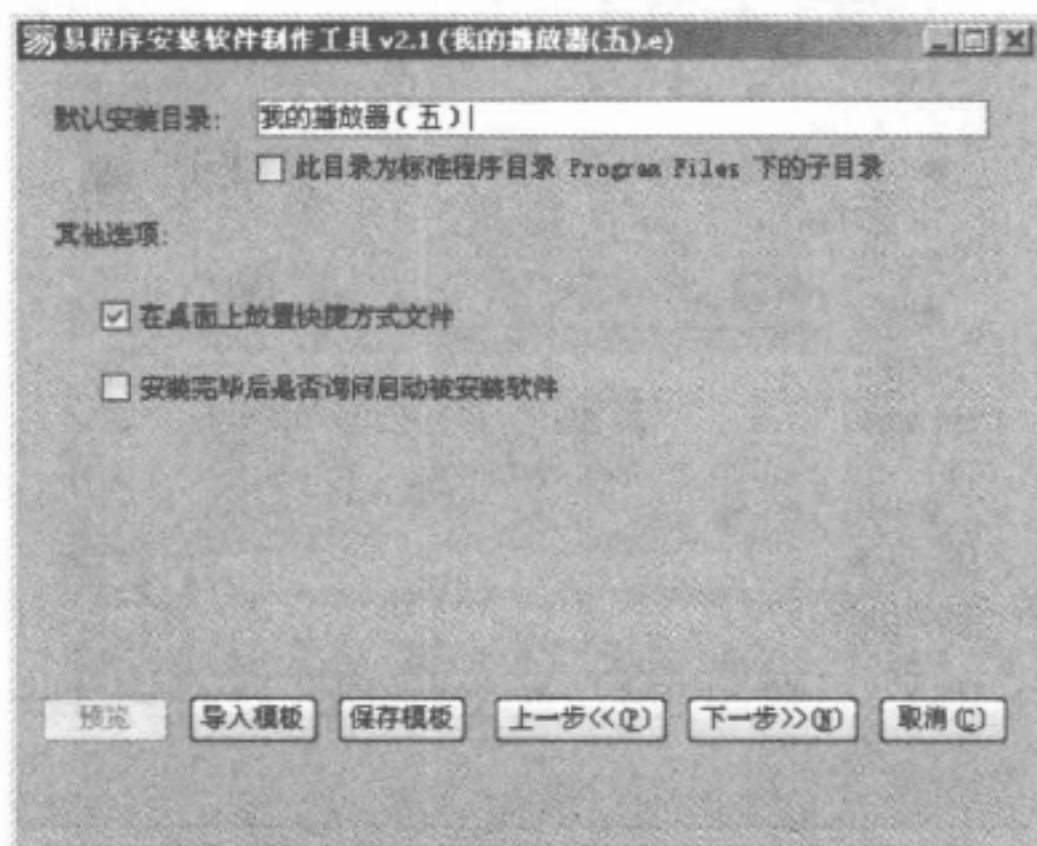


图15-32 设置软件安装时的默认安装目录

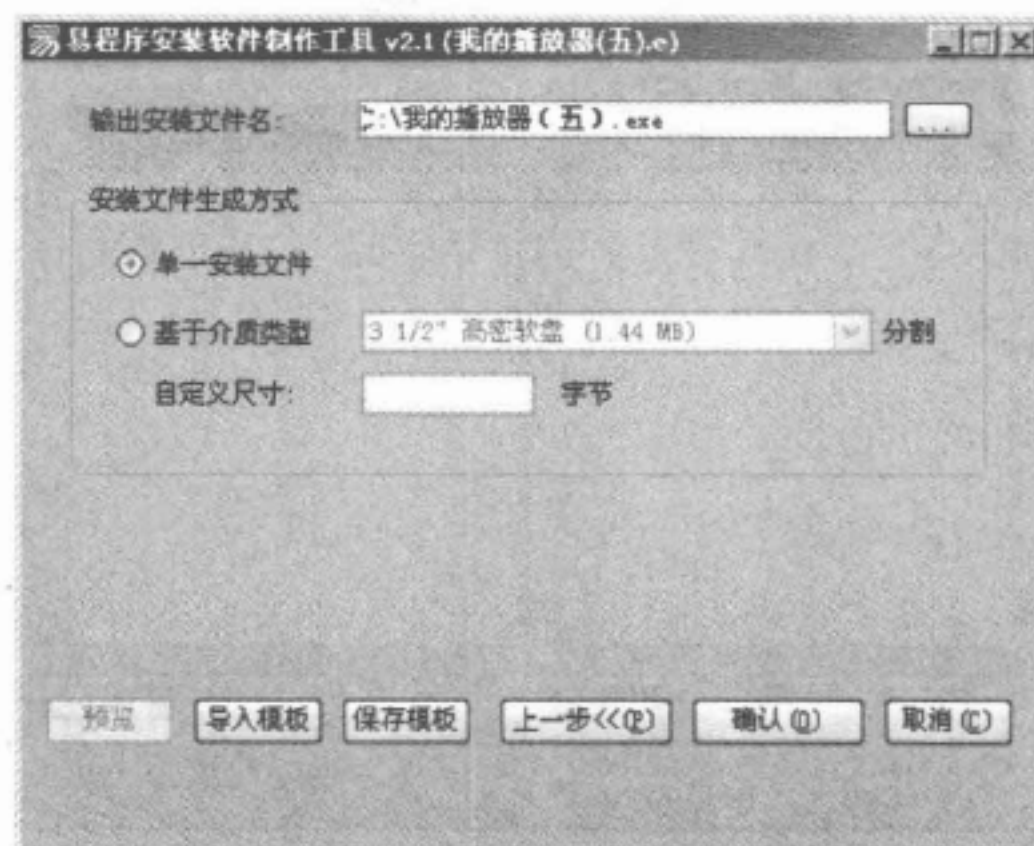


图15-33 选择保存制作后的安装文件的目录

编译安装完成之后，会弹出一个“生成安装软件成功”的提示窗口，如图15-34所示。安装向导编译所生成的安装文件使用了默认图标样式，如图15-35所示。



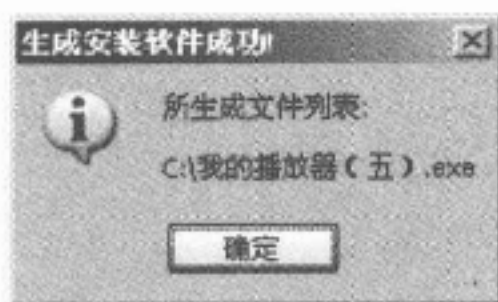


图15-34 提示生成安装软件成功



图15-35 生成的安装软件

安装文件制作完成，在系统C盘下就会看到保存的安装包文件了。

## 15.5 小结

易语言的5.X版本实现了静态编译。相对来说易语言只有两种编译方式：编译和独立（静态）编译。易语言的4.X版本和5.X版本，编译的方式是相似的，独立（静态）编译方式的不同之处在于使用的是静态支持库还是动态支持库。发布程序要将所有涉及到的文件打包，制作成安装文件方便用户使用。

## 15.6 习题

- 15-1 为保护目的程序的安全，易语言加入了\_\_\_\_\_和\_\_\_\_\_。
- 15-2 怎样设置程序的图标？
- 15-3 什么是静态编译？
- 15-4 易语言有哪些编译方式？
- 15-5 程序源代码的密码在哪里设置？




## 第五部分

# 易语言程序的解读和程序设计思路

通过前面的学习，相信大家对易语言提供的基础功能已经比较了解。但在自己真正编写或阅读他人的代码时，可能仍然觉得无从下手。这个问题在学习编程的人中是具有普遍性的，本部分通过带领大家解读和设计程序，来帮助读者尽可能地越过这个障碍。





## 第十六章 解读学校图书管理系统

### 本章目标

在本章结束时，我们能够：

- 学习简单的软件需求分析
- 解读易语言编写的管理软件
- 学习编程的程序设计思路





## 16.1 软件需求分析

软件需求分析所要做的工作是深入描述软件的功能和性能，设计软件的规则和软件同其他系统元素的接口细节，定义软件的其他有效性需求。

通过需求分析，逐步细化对软件的要求，描述软件要处理的数据域，并给软件开发提供一种可转化为数据设计、结构设计和过程设计的数据及功能表示方式。在软件完成后，制定的软件规格说明还要为评价软件质量提供依据。

软件需求分析是一个涉及面很广的学科，限于篇幅本书只进行简单的几项分析。软件需求分析在大型软件开发时是非常重要的一环，希望大家多寻找相关资料进行学习，相信会对以后的发展有很大的帮助。

### 16.1.1 软件使用环境

软件使用环境是指设计的软件要为哪些特定群体服务，要运行在哪些区域。

例如：某学校图书馆内有若干本藏书，为对该馆内藏书进行电子化管理，设计一套日常管理软件，该软件使用在图书馆局域网范围内各台式机、终端机，使用者为图书管理人员。

该学校图书馆现存图书已按照区域、书架、层数进行划分，并为每本书贴上一个拥有唯一ID的标签。

标签ID的格式为：区域 书架编号 层数 书在该层的序号

区域由1位英文字母组成

书架编号由2位英文字母组成

层数由2位数字组成

书序号由3位数字组成

标签ID例：AAB01001

标签表示：A区AB书架、1层、第1号位置

### 16.1.2 功能需求分析

以计算机软件替代使用纸、笔进行记录处理。

需要的基本功能有借书登记、还书处理、以指定条件进行记录查询、班级名称的管理、图书ID管理、有简单的管理员权限的分级。

#### 1) “借书登记”功能

需要记录借出书的ID、借书人姓名、所属年级和班级、以借出的时间计算还书最后日期。





## 2) “还书处理”功能

在已有记录中可以按一定的条件查找记录，并进行图书还回的处理。

## 3) “班级管理”功能

对软件内置的班年级进行查找和管理。

## 4) “图书管理”功能

对图书的ID、所对应的名称进行管理。

## 5) “账号管理”功能

对使用软件的人员根据登入账号的权限进行相应的功能限制，且有权限的账号可以对已有账号进行管理。

### 16.1.3 软件运行环境需求

硬件最低需求：

CPU: 500MHz

内存: 64MB

硬盘: 10MB剩余空间

外设: 标准键盘鼠标

软件最低需求：

Windows 95操作系统

## 16.2 解读学校图书管理系统

### 16.2.1 试运行程序

既然是阅读别人的代码，那么就先运行一下这个程序来看看实际的运行效果，如图16-1所示。

在出现的登入窗口中，为了测试程序时方便，默认填写了管理员用户名和密码。点击登入按钮进入管理程序，界面如图16-2所示。

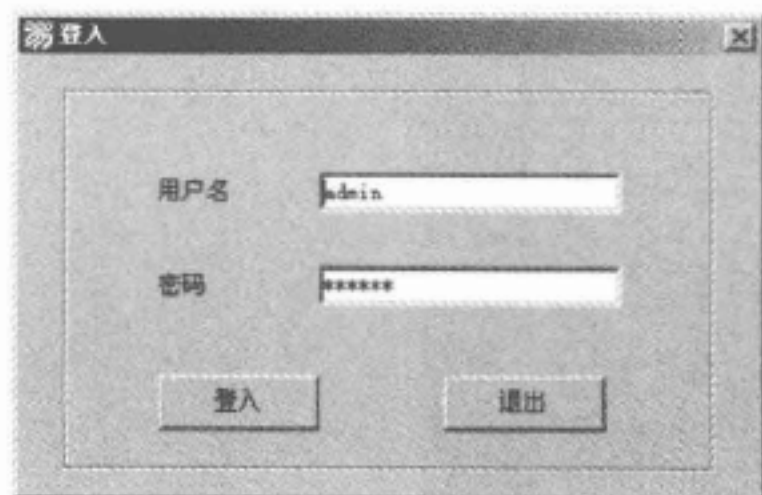


图16-1 登入窗口

图书ID	借书日期	还书日期	借书人	所属班级
AA123	2010年4月7日	2010年4月7日	ABC	一年一班
AA122	2010年4月7日	2010年4月10日	ABC	一年一班
AA121	2010年4月7日	2010年4月10日	ABC	一年一班
AA120	2010年4月7日	2010年4月14日	ABE	一年二班
AA121	2010年4月7日	2010年4月14日	ABB	一年二班
AA120	2010年4月7日	2010年4月14日	ABB	一年二班

图16-2 程序主界面





由于此时使用的是最高权限的账号，所以所有功能都可以使用。程序功能比较简单明确，可以模拟一些数据来试试各项功能。

## 16.2.2 解读程序

如果在阅读一个程序前，对该程序的功能没有一个简单的轮廓式的了解，或想完全阅读程序的每一部分，那么就应该跟着程序的运行流程来顺序阅读。

下面按照程序的运行流程来顺序解读“学校图书管理系统”。

**注解：**“学校图书管理系统”的源码在随书光盘“\图书例程\第十六章\学校图书管理系统\”中。

易语言窗口程序启动时最先执行的有两种方式：执行“\_启动子程序”和执行“\_启动窗口\_创建完毕”事件子程序。

具体是采用哪种方式，要看程序的系统配置。由于本程序以“\_启动子程序”的方式执行，所以从“\_启动子程序”开始解读，如图16-3所示。

“\_启动子程序”在“登入窗口程序集”中，代码如下：

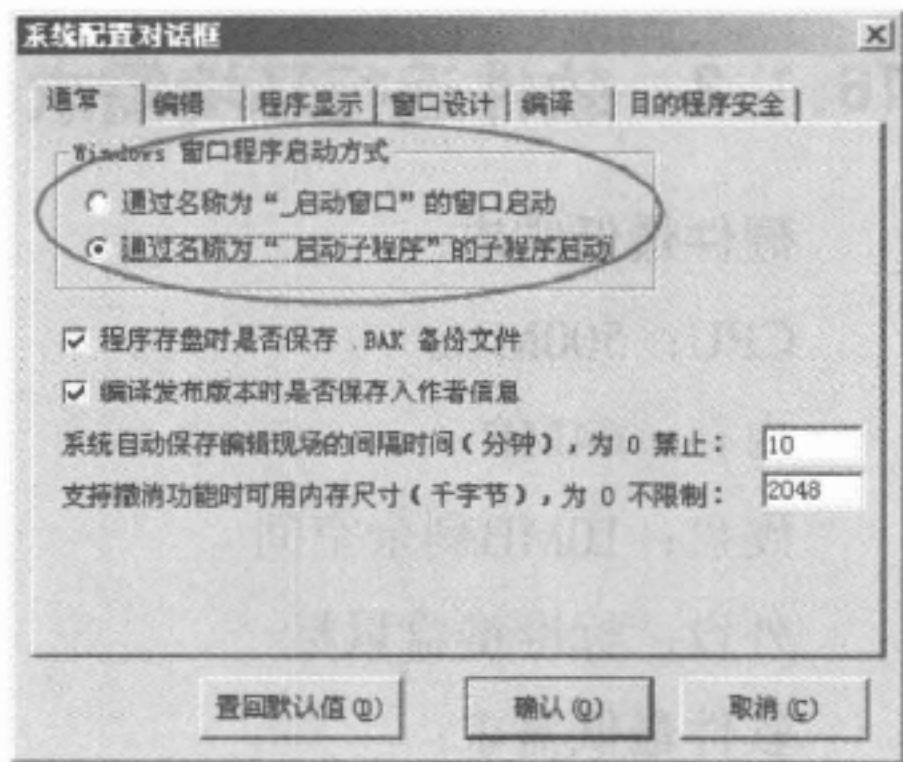


图16-3 系统配置

子程序名	返回值类型	公开	备注
_启动子程序	整数型		

```
当前管理员权限 = -1
运行目录 = 取运行目录 ()
--- 如果真 (取文本右边 (运行目录, 1) ≠ "\\")
    运行目录 = 运行目录 + "\\"
载入 (登入窗口, , 真)
--- 如果真 (当前管理员权限 > 0)
    载入 (主管理窗口, , 真)
返回 (0)
```

程序执行后，给两个变量进行了赋值，但我们并不知道这两个变量的定义。可以使用鼠标在变量调用处停留的方法来查看变量的信息，如图16-4所示。

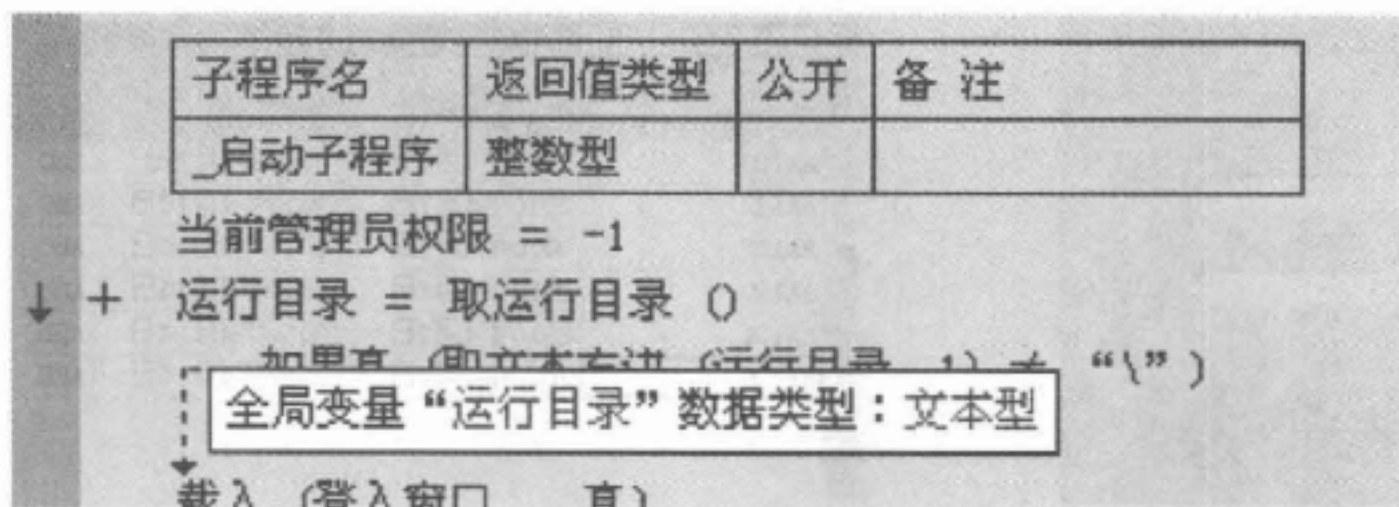


图16-4 查看变量信息





可以看到它是一个全局变量，我们到全局变量定义表中看看还有什么全局变量，如图16-5所示。

全局变量名	类型	数组	公开	备注
当前管理员权限	整数型			
运行目录	文本型			
今天日期	日期时间型			

图16-5 全局变量定义表

可以看到程序共定义了3个全局变量。除了刚才在“\_启动子程序”中见到的，还有“今天日期”变量。虽然从全局变量表中看到了变量的定义，但变量表中没有任何关于变量作用的说明性文字，只能通过变量名，对变量的作用进行一个简单的猜测（所以在编写程序的时候要养成添加注释的好习惯，这样对自己以后再次改写代码，或方便别人阅读都有帮助）。

回到“\_启动子程序”，继续来阅读代码。

子程序名	返回值类型	公开	备注
_启动子程序	整数型		

```

当前管理员权限 = -1
运行目录 = 取运行目录 ()
--- 如果真 (取文本右边 (运行目录, 1) ≠ "\")
运行目录 = 运行目录 + "\"
载入 (登入窗口, 真)
--- 如果真 (当前管理员权限 > 0)
载入 (主管理窗口, 真)
返回 ()
  
```

“当前管理员权限”被赋值为“-1”，我们现在并不知道其具体意义，这很正常。可以先不去理会它，继续阅读代码，在后面的代码中再去寻找答案。

下面是对“运行目录”的操作。这时就可以基本看出“运行目录”的作用了。它保存了程序当前的运行目录，并确保是由“\”结尾的。

接下来程序载入了“登入窗口”。由于是通过对话框方式载入的，所以下面的代码暂时不会被执行。我们跟着程序运行的流程走，跳到“登入窗口”的创建完毕事件。

子程序名	返回值类型	公开	备注
_登入窗口_创建完毕			

```

打开 (运行目录 + #管理员信息表 + ".edb", #管理员信息表, , , , )
  
```

只有一句代码，打开运行目录下的“管理员信息表”数据库。这里使用了“管理员信息表”常量，跳到常量表看看它的定义，如图16-6所示。

这里定义了一些文本型常量，根据名称和说明文字判断应该是对应的数据库名、别名和字段名。这个猜测是否正确还需要阅读程序后进行验证。

看过了“创建完毕”事件后，没有发现其他在“登入窗口”显示前执行的代码。接下来看看“登入窗口”中的组件都完成了哪些功能。





常量名称	常量值	公开	备注
班级信息表	“班级信息表”		班级信息表.edb
班级	“班级”		
班级类型	“班级类型”		1为年级、2为班级
借书登记表	“借书登记表”		借书登记表.edb
图书ID	“图书ID”		
借书日期	“借书日期”		
还书日期	“还书日期”		
借书人	“借书人”		
所属班级	“所属班级”		
图书信息表	“图书信息表”		图书信息表.edb
书名	“书名”		
类型	“类型”		
管理员信息表	“管理员信息表”		管理员信息表.edb
用户名	“用户名”		
密码	“密码”		
权限	“权限”		1最高权限、2无法管理帐户、3只能查看借出和图书信息

图16-6 常量定义表

登入窗口中有两个编辑框和两个按钮——“用户名编辑框”和“密码编辑框”。其中密码编辑框的输入方式属性设为“密码输入”方式，以便在运行时将输入的文字以“\*”代替。再来看看程序代码针对两个编辑框都做了哪些工作。

子程序名	返回值类型	公开	备 注		
_用户名编辑框_放开某键	逻辑型				
参数名	类 型	参考	可空	数组	备 注
键代码	整数值				
功能键状态	整数值				

--- 如果真 (键代码 = #回车键)  
 密码编辑框.获取焦点 ()

子程序名	返回值类型	公开	备 注		
_密码编辑框_放开某键	逻辑型				
参数名	类 型	参考	可空	数组	备 注
键代码	整数值				
功能键状态	整数值				

--- 如果真 (键代码 = #回车键)  
 \_登入按钮\_被单击 ()





程序分别处理了“用户名编辑框”和“密码编辑框”的放开某键事件。其中“用户名编辑框”事件中代码的功能是：在“用户名编辑框”中按下回车键时，将焦点移动到“密码编辑框”上；“密码编辑框”事件中代码的功能是：在“密码编辑框”中按下回车键时，执行“登入按钮”的“被单击”事件子程序中的代码。

可以看出上面的代码是在减少登入时鼠标的动作，使操作更快捷、方便。代码调用了“\_登入按钮\_被单击”子程序，程序如下：

子程序名	返回值类型	公开	备注
_登入按钮_被单击			

```

到首记录 ()
  如果 (查找 (读 (#用户名) = 用户名编辑框.内容) 且 读 (#密码)
    = 密码编辑框.内容)
    当前管理员权限 = 读 (#权限)
    登入窗口.销毁 ()
    信息框 ("用户名或密码错误", #错误图标 + #确认钮, "错误")
  
```

**注解：**在子程序调用处使用Ctrl+B，可以快速地跳转到该子程序的实现处。

在“\_登入按钮\_被单击”事件子程序中，程序首先将数据库的记录指针移到第一条记录。再在数据库中从头开始查找是否有用户名和密码都对编辑框内容的记录。如果有，读取该账号的权限，存放在“当前管理员权限”变量中，并销毁登入窗口；如果没有，则弹出一个信息框，提示错误。

到这里，解读的顺序是假设用户点击了“登入按钮”或在“密码编辑框”中按下了回车键。那么用户不做这些事，而是点击退出按钮或直接关闭登入窗口会怎么样呢？下面来看看剩下的两个事件子程序。

子程序名	返回值类型	公开	备注
_退出按钮_被单击			

```

登入窗口.销毁 ()

```

子程序名	返回值类型	公开	备注
_登入窗口_将被销毁			

```

关闭 (#管理员信息表)

```

“退出按钮”的“被单击”事件中，程序直接调用“登入窗口”的“销毁”命令。

“登入窗口”的“将被销毁”事件中，程序将已打开的数据库进行关闭。

到这里，整个登入窗口的代码都解读完了。那下一步程序该执行什么了？大家还记得在载入“登入窗口”的时候使用的是“对话框方式”吗？所以，这时候程序会去运行“载入”命令后面的代码。回到“\_启动子程序”，从载入“登入窗口”的位置开始向下看。





子程序名	返回值类型	公开	备注
_启动子程序	整数型		

```
当前管理员权限 = -1
运行目录 = 取运行目录 ()
-- 如果真 (取文本右边 (运行目录, 1) ≠ "\")
    运行目录 = 运行目录 + "\"
载入 (登入窗口, , 真)
-- 如果真 (当前管理员权限 > 0)
    载入 (主管理窗口, , 真)
返回 (0)
```

这时，就可以看出“当前管理员权限”的一个作用了——用来判断用户是否已经登入成功。在“\_启动子程序”一开始，就将它赋了一个“-1”。如果在“登入窗口”中，登入成功，它的值会是一个大于0的值；如果直接退出登入或直接关闭登入窗口，那么它的值不变，还是“-1”。程序就不必进入主管理界面，而是直接运行结束。到这，登入部分功能实现的代码已经看完了，其主要目的是对使用软件的人员作出简单的验证。

为了继续阅读全部程序，假设用户已经使用正确的账号登入了，我们跟着程序的流程继续向下看。

程序以对话框方式载入了“主管理窗口”，跟前面一样，我们跳到“主管理窗口”的“创建完毕”事件。

子程序名	返回值类型	公开	备注
_主管理窗口_创建完毕			

```
' 初始化
今天日期 = 取日期 (取现行时间 ())
打开 (运行目录 + #借书登记表 + ".edb", #借书登记表, , , ,
, )
打开 (运行目录 + #图书信息表 + ".edb", #图书信息表, , , ,
, )
打开 (运行目录 + #班级信息表 + ".edb", #班级信息表, , , ,
, )
打开 (运行目录 + #管理员信息表 + ".edb", #管理员信息表, ,
, , , )
```

```
' 更新表格数据
显示最近到时记录 (10, 真)
```

```
' 更新班级信息
更新班级列表 ()
```

```
' 根据帐号权限屏蔽功能
```

```
-- 如果真 (当前管理员权限 = 3)
    添加图书按钮.禁止 = 真
    删除图书.禁止 = 真
    还书.禁止 = 真
```

```
功能选择夹.现行子夹 = 1
```





事件子程序中，首先给全局变量“今天日期”赋值，它的值是现行系统时间的日期部分。

下面依次打开所需使用的数据库。

调用了一个自定义子程序“显示最近到时记录”，我们跟进这个子程序看看它做了哪些工作。

子程序名	返回值类型	公开	备 注		
显示最近到时记录			显示欲到时间还书的记录		
参数名	类 型	参考	可空	数组	备 注
最多显示条数	整数型				
显示已超时的记录	逻辑型				

变量名	类 型	静态	数组	备 注
循环变量	整数型			
字段数量	整数型			

' 清空当前显示记录的记录号组  
清除数组 (当前显示记录号组)

' 清空表格内容  
数据源.删除行 (1, 数据源.取行数 () )  
数据源.删除列 (1, 数据源.取列数 () )

“显示最近到时记录”没有返回值，有两个参数，分别是“最多显示条数”和“显示已超时的记录”。根据名称和说明推断，这个子程序的主要作用是显示最近快要到还书期限的记录。两个参数分别控制显示的记录最大条数和是否显示已超时尚未归还的记录。

子程序中定义了两个整数型变量，这里先不管它们的作用，继续向下看。

首先，程序清除了名为“当前显示记录号组”的变量。通过查看，它是一个程序集变量，定义如下：

窗口程序集名	保 留	保 留	备 注
主窗口程序集			
变量名	类 型	数组	备 注
当前显示记录号组	整数型	0	

回到“显示最近到时记录”子程序。程序接下来清空了表格的当前显示内容，估计是为显示新记录做准备。子程序下面代码如下：

```
' 排序借书登记表
置当前库 (#借书登记表)

如果真 (排序 (运行目录 + #借书登记表 + "new.edb", #还
书日期, , , ) = 真)
```



```

关闭 ()
删除文件 (运行目录 + #借书登记表 + ".edb")
文件更名 (运行目录 + #借书登记表 + "new.edb", 运行目录
+ #借书登记表 + ".edb")
打开 (运行目录 + #借书登记表 + ".edb", #借书登记表, , ,
, )

```

首先改变当前库，确保当前操作的是“借书登记表”。

接下来程序使用“排序”命令以“还书日期”为条件，对整个数据库排序。并使用排序后的新库替换原来的旧库。重新打开已排序的数据库。

```

' 向表格插入内容
置当前库 (#借书登记表)
字段数量 = 取字段数 ()

' 设置表头
数据源.添加行 (1)
数据源.置表头行数 (1)
数据源.插入列 (1, 字段数量)
--▶ 计次循环首 (字段数量, 循环变量)
    数据源.置文本 (1, 循环变量, 取字段名 (循环变量))
--- 计次循环尾 ()

```

这部分代码主要是为显示符合条件的记录做准备，将库中的字段名作为表头显示。这里“字段数量”的用处就可以看出来了。它保存了“借书登记表”库的字段数量，作用是作为循环的目标值使用。

```

到首记录 ()
--▶ 判断循环首 (尾记录后 () = 假 且 最多显示条数 > 0)
    如果真 (显示已过时的记录 = 假 且 今天日期 > 读 (#还书
    日期))
        跳过 (1)
        到循环尾 ()
    数据源.添加行 (1)

    --▶ 计次循环首 (字段数量, 循环变量)
        数据源.置文本 (数据源.取行数 (), 循环变量, 到文本 (读
        (循环变量)))
    --- 计次循环尾 ()

    加入成员 (当前显示记录号组, 取记录号 ())

    跳过 (1)
    最多显示条数 = 最多显示条数 - 1
    --- 判断循环尾 ()

```





接下来程序先确保了从数据库中的第一条记录开始操作。

使用判断循环首先将满足条件的记录显示在表格中。结束循环的条件有两个：当前的记录指针已经在尾记录后和已经循环了所需要显示记录的数量。

在循环内部，首先判断了当不需要显示已超时的记录时，当前记录的还书日期是已超时的，如果是则直接跳到下一条记录，并进入下一次循环。

如果上面的判断不成立，那么说明可能是以下情况之一：指定了需要显示已超时的记录和当前的记录还没有超时。无论是哪种情况，都符合需要显示的条件。下面程序开始将当前的记录内容显示在表格中。

首先在表格中添加了一个新行；使用循环将当前记录的各个字段数据填写到新加行的对应列中。

然后将当前记录号添加为“当前显示记录号组”的一个新成员，从这里就可以看出“当前显示记录号组”中，应该保存的是所有在表格中显示的记录在数据库中对应的记录号。

跳到数据库中的下一条记录，将“最多显示条数”减1，进入下一次循环。这样做就可以和判断循环首的条件对应起来形成完整的循环结束条件了。

“显示最近到时记录”子程序已看完了，回到“主管理窗口”的“创建完毕”事件中继续阅读。

```
' 更新表格数据
显示最近到时记录 (10, 真)

' 更新班级信息
更新班级列表 ()
```

程序又调用了一个自定义子程序，同样跟进查看。

子程序名	返回值类型	公开	备注
更新班级列表			

借书人年级组合框. 清空 ()  
借书人班级组合框. 清空 ()  
还书人年级组合框. 清空 ()  
还书人班级组合框. 清空 ()  
还书人年级组合框. 加入项目 (“(全部)”, )  
还书人班级组合框. 加入项目 (“(全部)”, )

该子程序无返回值，无参数。先阅读该子程序，然后再来总结它的作用。

程序首先将几个显示班、年级信息的组合框的项目清空，又向“借书人”的班、年级组合框中加入了一个默认项目。



置当前库 (#班级信息表)

到首记录 0

--> 判断循环首 (查找 (读 (#班级类型) = 1) = 真)

借书人年级组合框.加入项目 (读 (#班级), )

还书人年级组合框.加入项目 (读 (#班级), )

跳过 (1)

--- 判断循环尾 0

到首记录 0

--> 判断循环首 (查找 (读 (#班级类型) = 2) = 真)

借书人班级组合框.加入项目 (读 (#班级), )

还书人班级组合框.加入项目 (读 (#班级), )

跳过 (1)

--- 判断循环尾 0

接下来程序通过两个循环把两种类型的数据分别加入对应的年、班级组合框中。从这段代码中还可以看出“班级信息表”中“班级类型”为1代表的是年级名称，为2代表的是班级名称。

--- 如果真 (借书人年级组合框.取项目数 () > 0)

借书人年级组合框.现行选中项 = 0

--> 如果真 (借书人班级组合框.取项目数 () > 0)

借书人班级组合框.现行选中项 = 0

还书人年级组合框.现行选中项 = 0

还书人班级组合框.现行选中项 = 0

子程序的最后为各年、班级列表组合框选定一个默认项。

该子程序的代码看完了，通过阅读代码可以确定这个子程序的功能为从“班级信息表”中读取班级和年级的数据，在对应的组合框中显示。

回到“主管理窗口”的“创建完毕”事件，继续向下看。

’ 更新班级信息

更新班级列表 0

’ 根据帐号权限屏蔽功能

--- 如果真 (当前管理员权限 = 3)

添加图书按钮.禁止 = 真

删除图书.禁止 = 真

还书.禁止 = 真

功能选择夹.现行子夹 = 1

程序判断“当前管理员权限”是否为3（只能查看借出和图书信息通过登入代码和常量表说明判断出的），如果是，则通过禁止属性来限制可以使用的功能。看来“当前管理



员权限”变量的作用不单只是用来判断是否成功登入，还和功能限制息息相关。

将“功能选择夹”的当前显示的子夹切换到第二个子夹，即还书处理功能部分。

到这里程序后台的初始化工作就做完了，下面通过模拟使用各个功能所创造的程序流程来继续解读。为了方便，假设当前登入的账号拥有最高权限，没有任何功能限制。

先来看看针对用于功能切换的“功能选择夹”有哪些功能代码。

“功能选择夹”程序中处理了它的“子夹被改变”事件，我们从该事件子程序处开始阅读。

子程序名	返回值类型	公开	备注
_功能选择夹_子夹被改变			

```

--判断 (功能选择夹.现行子夹 < 2) ' 还回处理和借出登记
  如果真 (当前管理员权限 = 3 且 功能选择夹.现行子夹 = 0)
    信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
    功能选择夹.现行子夹 = 1
  --还书查询按钮_被单击 ()
  --判断 (功能选择夹.现行子夹 = 2) ' 班级管理
    如果真 (当前管理员权限 = 3)
      信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
      功能选择夹.现行子夹 = 1
      _还书查询按钮_被单击 ()
      返回 ()
  --班级查询按钮_被单击 ()
  --判断 (功能选择夹.现行子夹 = 3) ' 图书管理
  --图书查询按钮_被单击 ()
  --如果真 (功能选择夹.现行子夹 = 4) ' 帐号管理
    如果真 (当前管理员权限 > 1)
      信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
      功能选择夹.现行子夹 = 1
      _还书查询按钮_被单击 ()
      返回 ()
  --帐号查询按钮_被单击 ()
  
```

从大体上看，是以“现行子夹”属性来做一些相应的处理。我们按照顺序逐一分析。

```

--判断 (功能选择夹.现行子夹 < 2) ' 还回处理和借出登记
  如果真 (当前管理员权限 = 3 且 功能选择夹.现行子夹 = 0)
    信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
    功能选择夹.现行子夹 = 1
  --还书查询按钮_被单击 ()
  --判断 (功能选择夹.现行子夹 = 2) ' 班级管理
  
```





第一个判断是处理当前子夹是第一个或第二个时的情况。

首先判断当前登入的管理员是否为最低权限，且当前的子夹索引是0（即第一个子夹）。如果条件成立，则显示一个提示信息框并将当前子夹切换到第二个子夹上。最后调用“还书查询按钮”的“被单击”事件子程序。通过分析，这段代码的作用为限制没有权限的用户禁止切换到第一个功能（借出登记）的子夹；并且在切换到第一个子夹或第二个子夹后，等于按下了“还书查询按钮”。继续看完这个事件子程序，再去看按钮的事件处理子程序。

```

▶ 判断 (功能选择夹.现行子夹 = 2) ' 班级管理
├── 如果真 (当前管理员权限 = 3)
│   ├── 信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
│   ├── 功能选择夹.现行子夹 = 1
│   ├── _还书查询按钮_被单击 ()
│   └── 返回 ()
└── _班级查询按钮_被单击 ()
▶ 判断 (功能选择夹.现行子夹 = 3) ' 图书管理

```

第二个判断处理了当切换到“班级管理”功能子夹时的情况。

跟前面一样，处理了没有权限的情况；如果有权限则执行“班级查询按钮\_被单击”的事件子程序。

```

▶ 判断 (功能选择夹.现行子夹 = 3) ' 图书管理
├── _图书查询按钮_被单击 ()
└── 如果真 (功能选择夹.现行子夹 = 4) ' 帐号管理

```

第三个判断处理了切换到“图书管理”功能子夹时的情况。这里只调用了“图书查询按钮”的“被单击”事件子程序。

```

├── 如果真 (功能选择夹.现行子夹 = 4) ' 帐号管理
│   ├── 如果真 (当前管理员权限 > 1)
│   │   ├── 信息框 ("您所使用的帐号没有查看该信息的权限", 0, )
│   │   ├── 功能选择夹.现行子夹 = 1
│   │   ├── _还书查询按钮_被单击 ()
│   │   └── 返回 ()
│   └── _帐号查询按钮_被单击 ()
└──

```

最后一个判断处理了当切换到“账号管理”功能子夹时的情况。

跟前面一样，处理了没有权限的情况；如果有权限则调用“账号查询按钮”的“被单击”事件。





“功能选择夹”的“子夹被改变”事件子程序看完了。但其中调用的几个按钮的事件子程序中代码功能还不知道，下面就来看看上面调用过的各按钮事件子程序都做了哪些事。

先来看“\_还书查询按钮\_被单击”事件子程序：

子程序名	返回值类型	公开	备注
_还书查询按钮_被单击			

变量名	类型	静态	数组	备注
满足条件的记录号	整数型			
本次查询符合条件	逻辑型			
循环变量	整数型			
字段数量	整数型			
记录总数	整数型			

子程序内定义了几个变量，但都没有说明作用。还采取老办法，先看代码，然后再来推断它们的作用。

```

' 清空当前显示记录的记录号组
清除数组 (当前显示记录号组)

' 清空表格内容
数据源.删除行 (1, 数据源.取行数 () )
数据源.删除列 (1, 数据源.取列数 () )

置当前库 (#借书登记表)
记录总数 = 取记录数 ()
字段数量 = 取字段数 ()

' 设置表头
数据源.添加行 (1)
数据源.置表头行数 (1)
数据源.插入列 (1, 字段数量)
--> 计次循环首 (字段数量, 循环变量)
    数据源.置文本 (1, 循环变量, 取字段名 (循环变量))
--- 计次循环尾 ()

--- 如果真 (记录总数 < 1)
    返回 ()

```

看着这段代码是不是觉得很眼熟？对啦！这跟我们前面看过的“显示最近到时记录”子程序几乎是一样的。对相同的部分这里就不再赘述了，只说说不同的地方。这里多了一个“记录总数”的处理。“记录总数”中保存“借书登记表”中记录的数量。如果数据库



中没有记录，则直接返回，不向下继续运行。

下面是个代码很多的循环，我们来一段一段地看。

```

到首记录 0
--> 循环判断首 0
    满足条件的记录号 = 取记录号 0
    -- 如果真 (还书ID编辑框.内容 ≠ "")
        如果真 (查找 (读 (#图书ID) = 还书ID编辑框.内容) = 假)
            跳过 0
            到循环尾 0
    -- 如果真 (还书人姓名编辑框.内容 ≠ "")

```

循环外首先确保了从数据库中的第一条记录开始操作。

循环中，先将当前的记录号保存在“满足条件的记录号”变量中。在确保“还书ID编辑框”的内容不为空时，在数据库中查找“图书ID”字段的数据与“还书ID编辑框”内容相同的记录，如果没有则跳到下一条记录，跳到循环尾位置处进行再循环的判断。

```

-- 如果真 (还书人姓名编辑框.内容 ≠ "")
    本次查询符合条件 = 查找 (读 (#借书人) = 还书人姓名编辑框.内容)
    如果真 (本次查询符合条件 = 假 或 满足条件的记录号 ≠ 取记录号 0)
        如果 (满足条件的记录号 + 1 > 记录总数)
            跳出循环 0
        跳到 (满足条件的记录号 + 1)
        到循环尾 0
    -- 如果真 (还书人年级组合框.内容 ≠ "" 且 还书人年级组合框.现行选中项 > 0)

```

这段代码在确保了“还书人姓名编辑框”的内容不为空时，在数据库中查找“借书人”字段的数据与“还书人姓名编辑框”内容相同的记录并将查询结果保存到“本次查询符合条件”变量中。

接下来判断了前面的查询是否没找到，或是找到的记录已经不满足上一个查询条件；如果是这两种情况之一，那么进入接下来的代码块。

块中判断了满足前面查询条件的记录的下一条记录是否是库中最后一条记录；如果是，则跳出循环；否则跳到满足前面查询的记录的下一条记录，并将程序流程跳转到循环尾处。





```

--> 如果真 (还书人年级组合框.内容 ≠ "" 且 还书人年级组合框.现行选中项 > 0)
--> 本次查询符合条件 = 查找 (读 (#所属班级) ≈ 还书人年级组合框.内容)
--> 如果真 (本次查询符合条件 = 假 或 满足条件的记录号 ≠ 取记录号 ()
--> --> 如果 (满足条件的记录号 + 1 > 记录总数)
--> --> 跳出循环 ()
--> --> 跳到 (满足条件的记录号 + 1)
--> --> 到循环尾 ()
--> 如果真 (还书人班级组合框.内容 ≠ "" 且 还书人班级组合框.现行选中项 > 0)

```

这段代码在确保了“还书人年级组合框”的内容不为空，且“还书人年级组合框”的“现行选中项”大于0，即“还书人年级组合框”已选择的项不是“（全部）”（还记得在“更新班级列表”子程序中默认加入了它吗？）或未选择。

接下来的代码跟上段代码基本一样。只是查询的条件进行了改变，查找库中“所属班级”字段数据近似等于“还书人年级组合框”的内容。

```

--> 如果真 (还书人班级组合框.内容 ≠ "" 且 还书人班级组合框.现行选中项 > 0)
--> 本次查询符合条件 = 查找 (读 (#所属班级) = 还书人年级组合框.内容 + 还书人班级组合框.内容)
--> 如果真 (本次查询符合条件 = 假 或 满足条件的记录号 ≠ 取记录号 ()
--> --> 如果 (满足条件的记录号 + 1 > 记录总数)
--> --> 跳出循环 ()
--> --> 跳到 (满足条件的记录号 + 1)
--> --> 到循环尾 ()
--> 如果真 (还书剩余天数组合框.内容 ≠ "" 且 还书剩余天数组合框.现行选中项 ≠ 0)

```

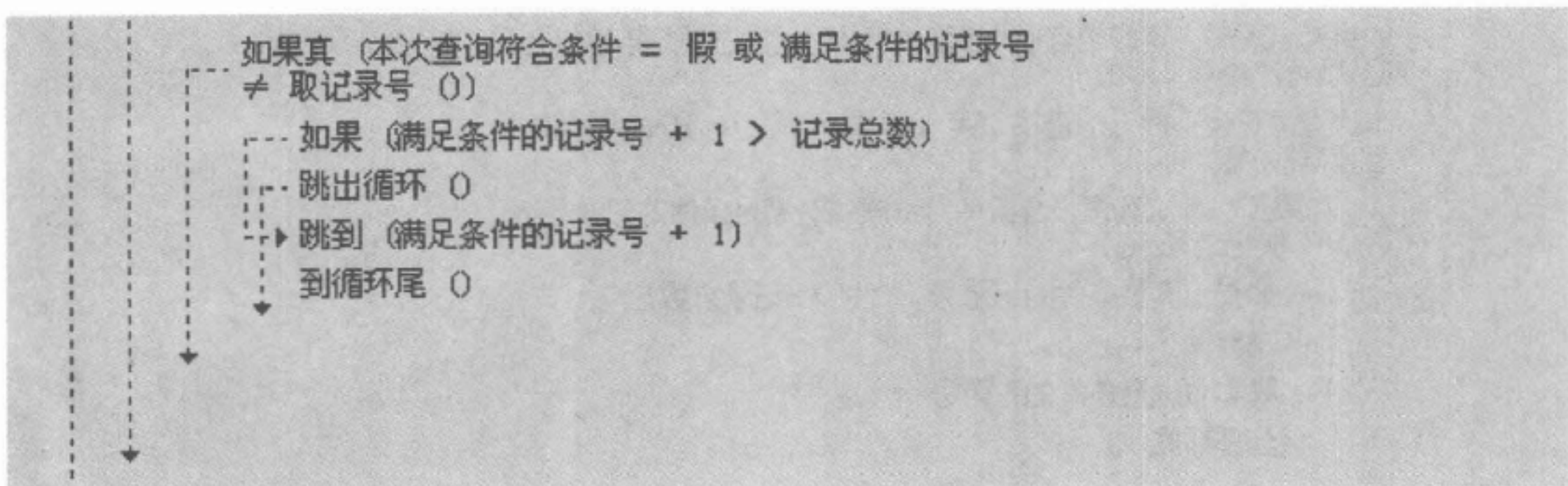
这段代码跟上段代码也很相似，只是改为“处理班级组合框”；查询的条件改为完整的年级、班级名。

```

--> 如果真 (还书剩余天数组合框.内容 ≠ "" 且 还书剩余天数组合框.现行选中项 ≠ 0)
--> --> 如果 (还书剩余天数组合框.现行选中项 = -1)
--> --> 本次查询符合条件 = 查找 (增减时间 (读 (#还书日期), #日, - 到数值 (还书剩余天数组合框.内容)) < 今天日期)
--> --> 本次查询符合条件 = 查找 (增减时间 (读 (#还书日期), #日, - 还书剩余天数组合框.取项目数值 (还书剩余天数组合框.现行选中项)) < 今天日期)

```





这段代码跟上段代码也很相似，判断部分只是改为处理剩余天数组合框。

但查询分了两情况：没有选择组合框的选项，即手动输入了内容和选择了组合框的选项。

如果是手动输入的内容，则查找条件为判断当前记录的“还书日期”字段日期减去输入的天数是否小于今天的日期。如果是，则说明当前的记录满足输入的指定还书天数内的条件。

如果不是手动输入的内容，条件一样。但所使用的减少天数的数值来自所选项的项目数值。“还书剩余天数组合框”的项目对应数值如图16-7所示。

对应的项名称如图16-8所示。

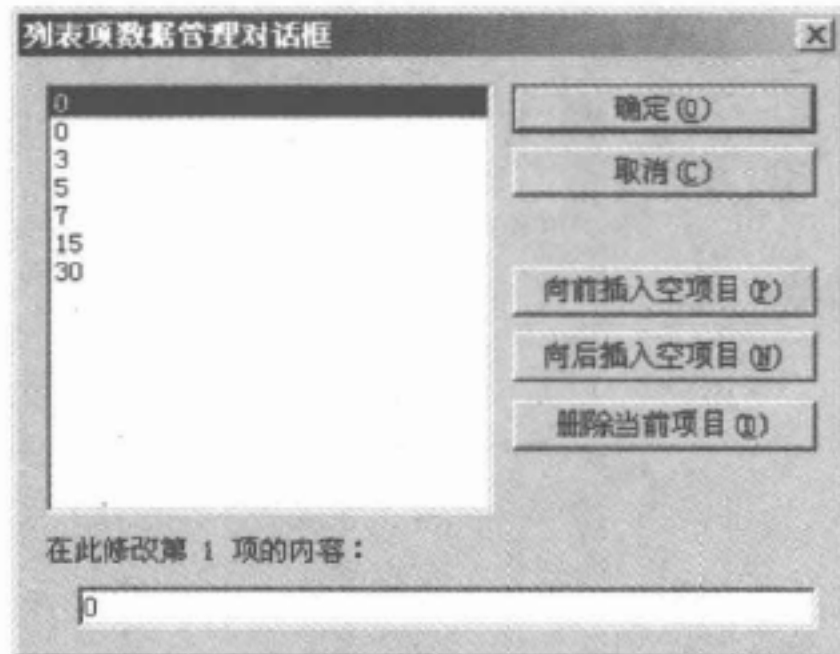


图16-7 还书剩余天数组合框项目数值

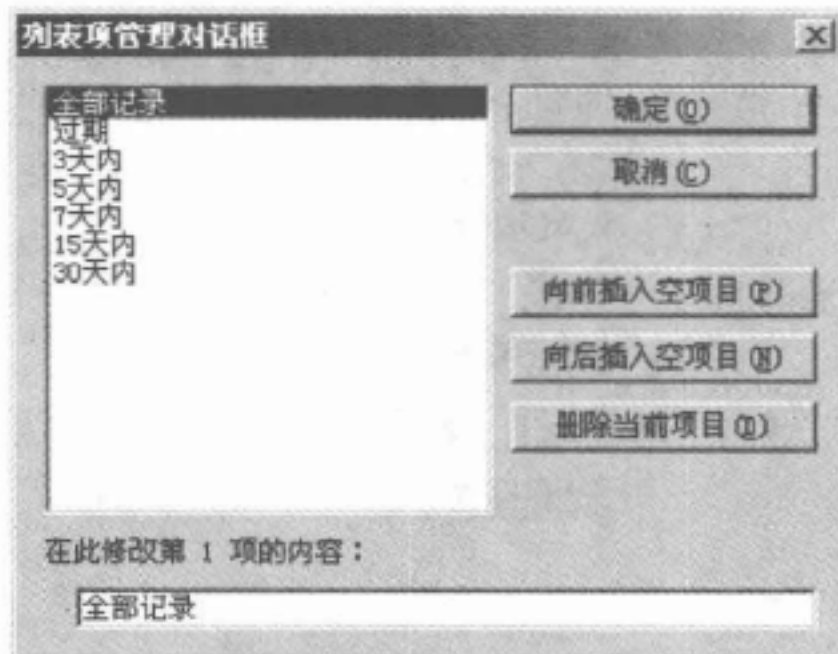


图16-8 还书剩余天数组合框项目名

```

    将满足条件的记录加入表格
    数据源.添加行 (1)
    -- 计次循环首 (字段数量, 循环变量)
    -- 数据源.置文本 (数据源.取行数 (0), 循环变量, 到文本 (读
    -- (循环变量)))
    -- 计次循环尾 (0)
    加入成员 (当前显示记录号组, 取记录号 (0))
    跳过 (0)
    -- 循环判断尾 (尾记录后 (0) = 假)
  
```

最后，将满足条件的记录显示在表格中，并记录在表格中显示的记录号。循环判断尾的循环条件是直到数据库当前记录指针在尾记录后。





到这整个事件子程序就读完了，下面来总结一下它的功能和各变量的作用。

这个事件子程序实现了对“借书登记表”的多条件可选式的查询，并将满足条件的记录显示在表格中。

各局部变量的作用如下：

满足条件的记录号：用于判断查询后，满足条件的记录是否满足前一个条件；用于改变数据库记录指针跳到正确的记录。

本次查询符合条件：用于判断某个查询的结果。

循环变量：作为各个循环的临时变量。

字段数量：用来作为读取字段名和各字段数据的循环目标值。

记录总数：用来判断当前库中是否有记录。

在“\_功能选择夹\_子夹被改变”的事件子程序中调用过的“\_班级查询按钮\_被单击”、“\_图书查询按钮\_被单击”、“\_帐号查询按钮\_被单击”事件子程序的功能和实现思想与刚才的“\_还书查询按钮\_被单击”事件子程序大同小异，所以不再赘述。

下面来看看其他几个按钮的功能。先从“\_借出登记按钮\_被单击”事件子程序看起。

子程序名	返回值类型	公开	备注
_借出登记按钮_被单击			

检查是否填写了所需信息

如果真 (借书ID编辑框.内容 = "" 或 借书人姓名编辑框.内容 = "" 或 借出天数组合框.现行选中项 = -1 或 借书人年级组合框.现行选中项 = -1 或 借书人班级组合框.现行选中项 = -1)

信息框 ("请填写信息后再添加", 0, )

返回 0

置当前库 (#借书登记表)

加空记录 0

写 (#图书ID, 借书ID编辑框.内容)

写 (#借书日期, 今天日期)

写 (#还书日期, 增减时间 (今天日期, #日, 借出天数组合框.取项目数值 (借出天数组合框.现行选中项)))

写 (#借书人, 借书人姓名编辑框.内容)

写 (#所属班级, 借书人年级组合框.取项目文本 (借书人年级组合框.现行选中项) + 借书人班级组合框.取项目文本 (借书人班级组合框.现行选中项))

显示最近到时记录 (10, 真)

首先，判断了登记时所需的数据是否都已填写，如果有一项没有填写，则弹出提示信息，并返回子程序，不再继续向下执行。

如果所需的信息都已填写，则在“借书登记表”中加入新的记录。将各组件的内容（有些经过简单的处理）写到对应的字段中。







调用“显示最近到时记录”子程序，显示10条并包括过期的记录。

这个事件子程序主要就是对数据库添加记录的处理，通过前面章节的学习，相信大家不难理解“\_添加班级按钮\_被单击”、“\_添加图书按钮\_被单击”、“\_添加帐户按钮\_被单击”事件子程序的功能都很类似，只不过是操作的表和数据不同。这里就不再赘述。

了解了“查询”、“添加”的操作，下面来看看“删除”部分的操作——“\_还书\_被选择”、“\_删除班级\_被选择”、“\_删除图书\_被选择”、“\_删除帐号\_被选择”事件子程序（都是菜单的响应事件）。简单看一下它们也很“雷同”，所以下面只解读一下“\_还书\_被选择”事件子程序。我相信大家都是很聪明的，可以做到“举一反三”。

子程序名	返回值类型	公开	备注
_还书_被选择			

变量名	类型	静态	数组	备注
选择行数	整数型			
循环变量	整数型			
光标行号	整数型			

```
选择行数 = 数据显示表格.取选择行数 ()
--- 如果真 (选择行数 < 1)
    信息框 (“请选择欲操作的记录”, 0, “没有选择记录”)
    返回 ()
-- 如果真 (信息框 (“是否要真的删除这些记录”, #是否钮, “还书
确认”) = #是钮)
    光标行号 = 数据显示表格.取光标行号 ()
    置当前库 (#借书登记表)
    变量循环首 (光标行号 + 选择行数 - 2, 光标行号 - 1,
-1, 循环变量)
    跳到 (当前显示记录号组 [循环变量])
    删除 ()
    --- 变量循环尾 ()
    删除成员 (当前显示记录号组, 光标行号 - 1, 选择行数)
    彻底删除 ()
    数据源.删除行 (光标行号, 选择行数)
```

代码很少，一眼就可以看出三个局部变量的作用。

选择行数：保存着“数据显示表格”中当前被选择的行数。

循环变量：用作循环时记录临时值的临时变量。

光标行号：保存着“数据显示表格”中当前光标所在的行号。

首先判断是否已选择了“数据显示表格”的内容；如果没有选择则弹出提示，并返回子程序不再向下执行。

如果选择了内容，弹出信息框让用户确认是否真的要删除；如果选择了“否”，则不





做任何事；选择“是”则开始删除所选内容对应的记录。

首先获取当前光标所在行号；改变当前库为“借书登记表”；使用循环为所有被选择的内容所对应的记录加上“删除标记”。由于表格有一个表头，所以这里使用“光标行号+选择行数-2”作为起始循环变量值；也是因为有表头，所以循环变量的目标值为“光标行号-1”；循环每次递减1；这个循环主要是用来控制“循环变量”来作为“当前显示记录号组”的索引，所以要注意表格的行号和“当前显示记录号组”的索引差1的问题。

删除“当前显示记录号组”内已被选择删除内容所对应的记录号成员；彻底删除数据库中被加上“删除标记”的记录；删除表格内被选择的内容。

到这里我们发现“当前显示记录号组”是为了在删除记录时，可以快速地找到表格内容所对应的数据库中记录号。前面读程序留下的各种“悬念”也一一解开了。

最后整个程序只剩下“\_数据显示表格\_鼠标右键被放开”和“\_主管理窗口\_将被销毁”事件子程序了。

子程序名	返回值类型	公开	备注		
_数据显示表格_鼠标右键被放开	逻辑型				
参数名	类型	参考	可空	数组	备注
横向位置	整型				
纵向位置	整型				
功能键状态	整型				

```

判断 (功能选择夹.现行子夹 < 2)
弹出菜单 (还回处理, )
判断 (功能选择夹.现行子夹 = 2)
弹出菜单 (班级管理, )
判断 (功能选择夹.现行子夹 = 3)
弹出菜单 (图书管理, )
如果真 (功能选择夹.现行子夹 = 4)
    弹出菜单 (帐号管理, )

```

“\_数据显示表格\_鼠标右键被放开”事件子程序，主要是根据当前的功能弹出对应的菜单。

子程序名	返回值类型	公开	备注
_主管理窗口_将被销毁			

全部关闭 0

“\_主管理窗口\_将被销毁”事件子程序中，将所有已打开的数据库进行关闭，来保证数据的安全性。







## 16.3 后记

在学习完一个程序的源码后，好的习惯是对这个程序的优、缺点进行总结。


“学校图书管理系统”值得大家学习的地方主要在功能的分类、操作的便利性，特别是“多条件可选择”查询的实现。

这个程序的缺点在于功能比较简单、数据关联性不强、使用的数据库系统不够强大，随着数据量的增多，程序性能会有较大的下降。

“学校图书管理系统”是一个典型的管理程序，也是典型的面向过程编程思想的程序。通过对它的解读，希望大家对一个现存源程序怎样学习有一个简单的了解。

学习软件的需求分析，在自己写程序时可以帮助大家细分问题，并对整个软件的结构有个大体的认识。要试着根据源码去做它的需求分析，这也是锻炼编写软件、解决实现思路的一个好途径。





## 第十七章 解读游戏——对对碰

### 本章目标

在本章结束时，我们能够：

- 学习简单的游戏策划分析
- 解读易语言编写的游戏软件
- 学习面向对象编程的程序设计



## 17.1 游戏策划分析

游戏的策划是整个游戏的开发蓝本，其在游戏开发中的地位与软件开发中需求分析的地位相同甚至更高。游戏的策划应在实际编写游戏前完成。游戏策划贯穿从图形到程序再到内容等所有细节，有时游戏的营销和市场的分析也归为此项。本章的案例相对比较简单，只做一些必要的分析。

### 17.1.1 游戏策划文档实例

游戏名称：对对碰

游戏类型：益智游戏

游戏载体：光盘和网络下载

硬件最低需求：

CPU：500MHz

内存：64MB

硬盘：10MB剩余空间

显卡：可支持16位色深并拥有1M显存

显示器：可显示16位色深

外设：标准键盘鼠标

软件最低需求：

Windows 95操作系统

游戏主要操作方式：

键盘、鼠标

游戏主界面结构草图（见图17-1）：

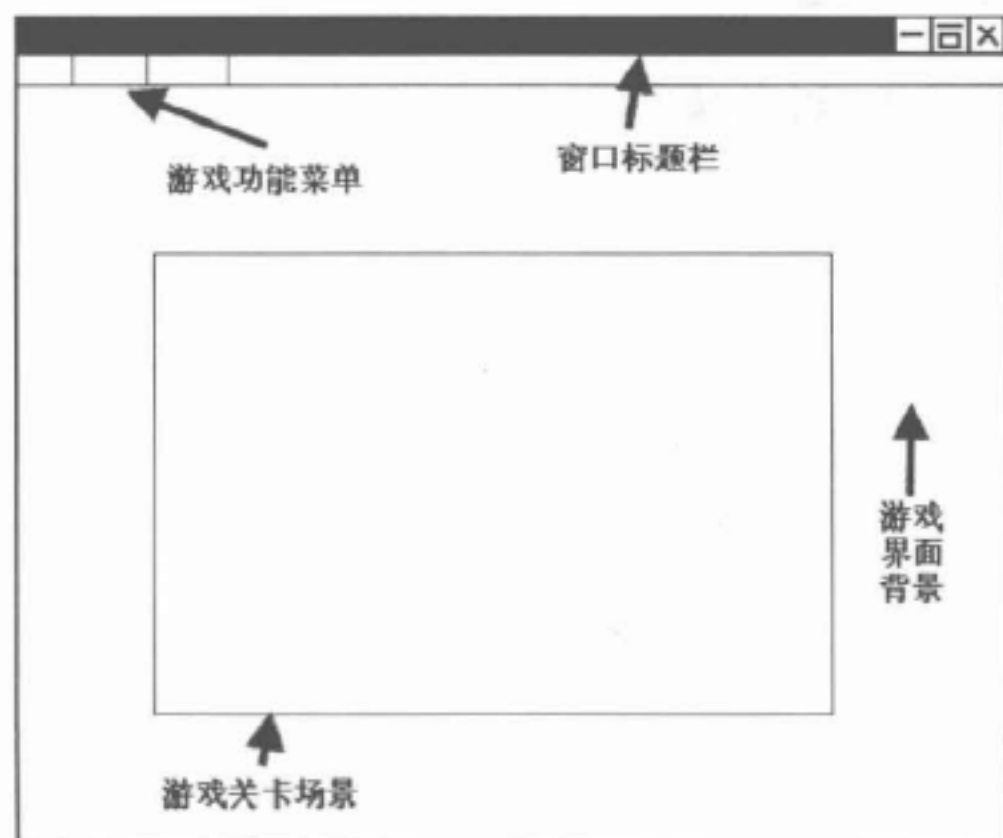


图17-1 游戏主界面结构草图





补充说明：界面中窗口最大化不可用。

功能菜单：

游戏（G）

重新开始（R）功能：重新开始本关

.....

跳到下一关（N）功能：跳到下一关游戏

跳到指定关（J）功能：跳到用户指定的关数

.....

退出（X）功能：退出游戏

帮助（H）

游戏帮助（H）功能：打开“记事本”显示游戏帮助文件

.....

关于（A）...功能：打开关于窗口

关于界面结构草图（见图17-2）：

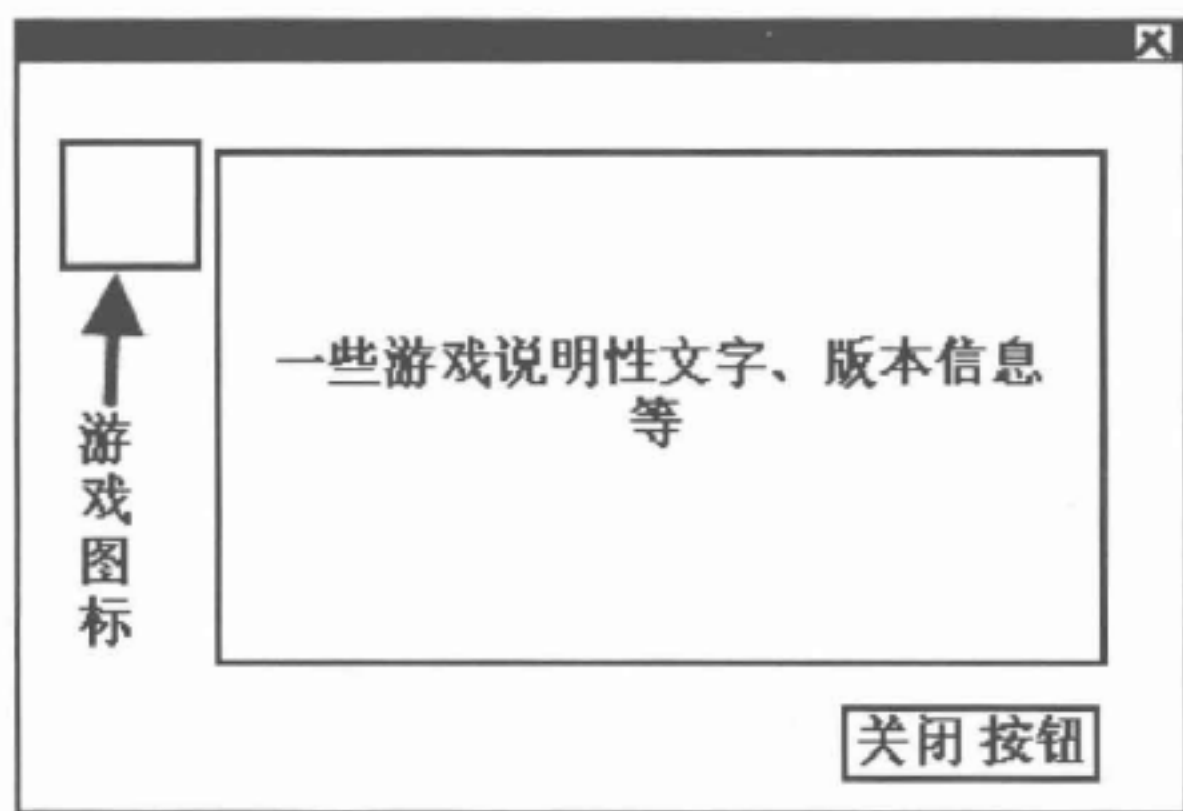


图17-2 界面结构草图

游戏玩法和规则：

游戏有两种类型的图块，一种是普通块，一种是特殊块。玩家通过移动普通的图块使得两个或两个以上相邻（4个方向，上、下、左、右）时，播放一段过渡动画然后一起消除。玩家的目标就是想办法将关卡中的所有普通块全部消除，即过关。普通块当下方为空时会自动下落，且玩家不能直接向上移动普通块。普通块的样式初期定为8种且为静态，程序需要预留更多种类和可动态块的扩展空间。

特殊块根据种类不同其功能也不同。初期定为4种特殊块，它们的具体功能如下。

① 墙：静态块且不可移动，不会自由下落，通常作为关卡的框架。

② 石头：静态块可以像普通块一样由用户进行移动，且亦会自由下落。但石头相邻时不会消除。





③ 火焰：动态块（两帧，循环播放的动画）且不可移动，不会自由下落。会对由火焰块上方落下的普通块和石头块产生影响。当由上方落到火焰块上的是普通块，则本关游戏失败；若由上方落到火焰块上的是石头块时，石头块消失，游戏继续。

④ 电梯：静态块，不可由用户控制移动，其会按自身类型横向或纵向移动，当不能继续向运动方向移动，则向相反方向移动，如此循环。电梯移动时，会带动其上的块（包括块上面的所有用户可移动的块）按其运动方向移动。特殊情况是，纵向电梯带着石头向上移动时，如果石头上有其他块将不能移动。

#### 游戏操作：

上移选择框：上光标键、数字键8

下移选择框：下光标键、数字键2

左移选择框：左光标键、数字键4

右移选择框：右光标键、数字键6

选中/取消：回车键、数字键5

重玩本关：F2键

跳到下一关：F3键

跳到指定关：Shift+F3键

#### 美工需求：

美工的風格需要帶有強烈的卡通色彩

(1) 8種不同的普通塊及其從完整到消失過程的動畫

圖片尺寸：20×20像素

格式：BMP

動畫幀數：4幀（每幀都是單獨文件）

透明色：品紅（H. FF00FF）

(2) 4種特殊塊

① 牆：主色調明亮，可以跟其他塊明顯區分，填滿20×20像素。

② 石頭：灰色為主，體現石塊的堅實和質量，填滿20×20像素。

③ 火焰（2幀）：鮮亮的火焰，類似岩漿式的流動；整尺寸20×20；透明色：品紅（H. FF00FF）。

④ 電梯：上下底結實有力；整尺寸20×20；透明色：品紅（H. FF00FF）。

(3) 選擇框

圖片尺寸：38×26像素

格式：BMP

透明色：藍色（H. FF0000）

未選擇狀態：內斂的、鏤空的正方形，正方形內塊尺寸正好為20×20。





选择状态：在上面的正方形基础外做左右方向的扩展，暗示用户可以左右移动。正方形内块尺寸正好为20×20。

关卡设计：

草图中代图形意义（见图17-3）：

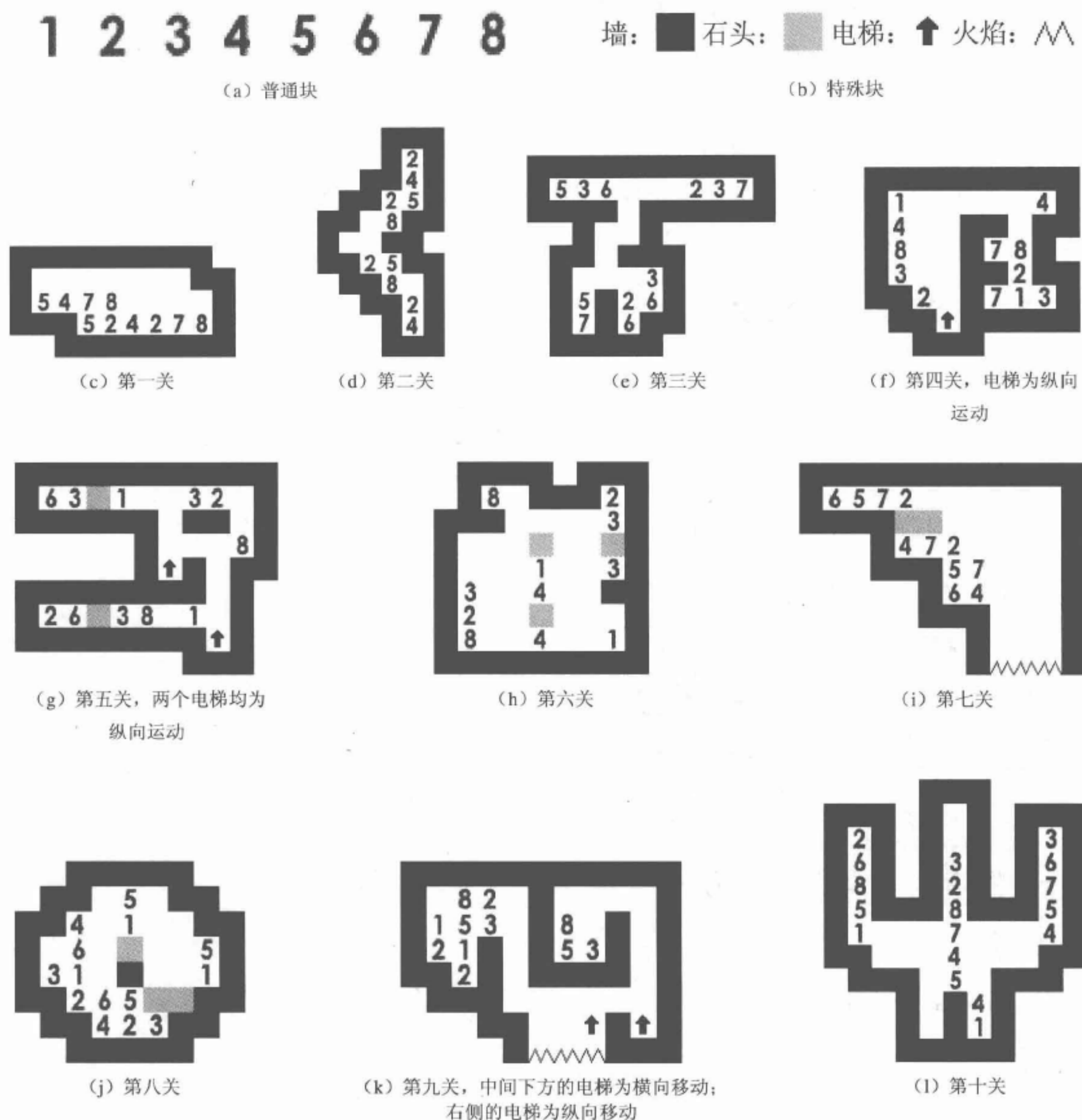


图17-3 草图中代图形

## 17.1.2 小结

上面我们看过了“对对碰”游戏的策划文档。其中主要包括美工和程序两方面的工作。在游戏策划完成后，程序和美工就可以按照策划文档进行各自的工作了。





在大多数情况下，从网上获得代码时并不会附带策划文档。但我们可以通过阅读源代码和试验程序的功能来反向分析出策划文档。这样做有助于开发过程中对程序各功能进行分析、细化时，有更清晰的思路。

## 17.2 解读游戏——对对碰

在阅读源代码时，大多数的情况是代码量很多，但只对其中的某一个功能或某一部分代码感兴趣。对于这样的情况，该怎样来阅读呢？下面就以“对对碰”游戏为例，通过对其关键部分的解读，来解决这个问题。

### 17.2.1 解读游戏构架

游戏程序通常都有一个“GameLoop”，即游戏循环。这个循环的生命周期从游戏初始化后开始，到游戏结束前终止。游戏循环主要有两方面工作：处理游戏中的逻辑和各元素的位置和更新游戏画面以显示游戏中各元素的最新状态。

在“对对碰”中，游戏循环使用时钟组件来实现，下面来看一下它的代码：

子程序名	返回值类型	公开	备注
_时钟1_周期事件			

变量名	类型	静态	数组	备注
启动时间	整数型			
耗费时间	整数型			

```
时钟1.时钟周期 = 0
--- 如果真 (游戏结束 = 真)
    信息框 (“游戏失败, 按F2重来”, #确认钮 + #信息图标, “失败”)
    返回 0

启动时间 = 取启动时间 ()
' 耗时代码
游戏逻辑 ()
重绘画面 ()
_画板1_绘画 (0, 0, 画板1.宽度, 画板1.高度)

' 时间计算
耗费时间 = 取启动时间 () - 启动时间
--- 如果 (耗费时间 < 周期长度)
    时钟1.时钟周期 = 周期长度 - 耗费时间
    时钟1.时钟周期 = 1
```



这里简单说明计算时间的作用。因为游戏的每秒刷新帧数·(FPS)一般是固定的，但每次刷新时的运算所消耗时间（一般为游戏逻辑和更新画面所消耗的）并不相同。所以要想保证固定的FPS，就需要进行时间补偿。上面这段程序使用执行耗时代码前和耗时代码后的启动时间差来对下次循环延时时间做补偿。

### 17.2.2 解读游戏逻辑

接下来详细解读整个游戏的重中之重——游戏逻辑子程序。

**注解：**由于代码较多，将采用分段讲解。

子程序名	返回值类型	公开	备注
游戏逻辑			

变量名	类型	静态	数组	备注
循环变量1	整型			
循环变量2	整型			
循环变量3	整型			
消除节点组	我的节点		0	
临时节点	我的节点			
临时行	整型			
临时列	整型			
当前节点索引	整型			
临时数据	整型			

更新特殊块数据

变量循环首 (取数组成员数 (特殊块组), 1, -1, 循环变量1)

特殊块组 [循环变量1].更新状态 ()

变量循环尾 ()

程序首先调用了当前各特殊块对象的“更新状态”方法。“更新状态”的作用，在后面再详细讲解。

块消除及下落处理
<div> <div>如果真 (移动中 = 真)</div> <div> <div>临时数据 = 0</div> <div>移动中 = 假</div> <div> <div>变量循环首 (取数组下标 (地图数组, 1), 1, -1, 循环变量1)</div> <div> <div>变量循环首 (取数组下标 (地图数组, 2), 1, -1, 循环变量2)</div> <div> <div>如果真 (移动中 = 真 且 取数组成员数 (消除块动画组) &gt; 0)</div> <div>判断是否有在播放消除动画的, 如果有先不下落</div> <div>返回 0</div> </div> </div> <div> <div>如果真 (循环变量1 &gt; 1)</div> <div>临时数据 = 地图数组 [循环变量1 - 1] [循环变量2]</div> </div> </div> </div> </div>





首先确定当前是否有块移动了，如果没有块移动则不会发生下落和需要消除的情况，加上这个判断可以使程序避免不必要的运算。接下来的两层循环的作用是遍历“地图数组”中的每一个成员，之所以从后向前遍历数组的各成员，其目的在于这样可以顺序地处理由于下面的块消除或下落后所连带引发的一系列判断需求。“地图数组”是一个二维整数型数组，其中成员为正值代表普通块的类型，负值代表“特殊块组”的索引，0代表这个位置没有块。它的初始化过程在“读地图”子程序中，由于该子程序没有什么难懂的算法，所以本书不再解读。

在循环中，首先判断是否有块在移动且在播放动画效果，如果是则跳出子程序。为什么在循环一开始就做这样一个判断呢？在循环外刚刚做了是否在移动中的判断，是不是再判断没有意义呢？其实有这些疑问很正常。这是由于我们不了解整个循环的工作过程，当我们阅读完整个这部分功能后，就会有豁然开朗的感觉。

接下来程序取得了“当前格”的上一格数据保存在“临时数据”变量中。

```

' 下落处理
' 判断条件：
' 判断当前格上边是否有格(没有超出数组边界)
' 判断当前格是否是空的
' 判断当前格上边是否是普通格或是可移动的特殊格
  如果 (循环变量1 - 1 > 0 且 地图数组 [循环变量1] [循环变量2] = 0
  且 (临时数据 > 0 或 临时数据 < 0 且 特殊块组 [-临时数据].是否禁止 (0 = 真))
  移动中 = 真 ' 让程序在下次循环时继续判断

' 对调数据
地图数组 [循环变量1 - 1] [循环变量2] = 0
地图数组 [循环变量1] [循环变量2] = 临时数据

  如果真 (选择框对象.取状态 (0) = 2 且 选择框对象.取所在行 (0) =
  循环变量1 - 1 且 选择框对象.取所在列 (0) = 循环变量2)
  ' 更新选择框位置
  选择框对象.置所在行 (循环变量1)
  选择框对象.置所在列 (循环变量2)
  选择框对象.置横坐标 (地图.取单元格横坐标 (选择框对象.取所在列 (0)))
  选择框对象.置纵坐标 (地图.取单元格纵坐标 (选择框对象.取所在行 (0)))

' 特殊块处理
  如果真 (临时数据 < 0)
  特殊块组 [-临时数据].置所在行 (循环变量1)

  临时数据 = 地图数组 [循环变量1] [循环变量2]
  
```





代码的注释对判断的解释比较清楚，下面看看满足下落条件后都做了些什么。首先将移动中的状态设为“真”，让程序在下一次判断中继续判断是否由于本次的处理又引发了其他的下落或消除块。接下来通过对调“当前格”和“当前格”上面的数据实现下落的效果。接下来是处理如果选择框当前选择的是所操作的下落块，则更新选择框的位置，实现跟随下落的效果。最后，如果下落的块是一个特殊块，那么需要更新一下该特殊块对象所对应的行数，以避免在后面使用时发生数据不对称的情况。

接下来看看块消除的处理代码：

```

--▶ 临时数据 = 地图数组 [循环变量1] [循环变量2]
    --- 如果真 (移动中 = 假 且 临时数据 > 0)
        ' 消除检查、连锁消除
        临时节点.行 = 循环变量1
        临时节点.列 = 循环变量2
        加入成员 (消除节点组, 临时节点)
        --▶ 判断循环首 (取数组成员数 (消除节点组) > 0)
            当前节点索引 = 取数组成员数 (消除节点组)
            临时节点 = 消除节点组 [当前节点索引]
    
```

“临时数据”保存了“当前格”的数据，并检测当前是否没有移动和“当前格”是否是普通块。以当前格为基准点，为探寻相同块做准备。在判断循环内，始终以当前“消除节点组”中的最后一个成员作为新一次探寻相同块的起始点。

```

--▶ 计次循环首 (4, 循环变量3)
    ' 判断当前块的4个方向相邻块是否与当前块相同
    --- 判断 (循环变量3 = 1)
        ' 左
        临时行 = 临时节点.行
        临时列 = 临时节点.列 - 1
    --▶ 判断 (循环变量3 = 2)
        ' 上
        临时行 = 临时节点.行 - 1
        临时列 = 临时节点.列
    --▶ 判断 (循环变量3 = 3)
        ' 右
        临时行 = 临时节点.行
        临时列 = 临时节点.列 + 1
    --▶ 判断 (循环变量3 = 4)
        ' 下
        临时行 = 临时节点.行 + 1
        临时列 = 临时节点.列
    
```

以当前节点为基准，分别生成四个方向的对应行列号。



```

如果 (临时行 < 1 或 临时行 > 取数组下标 (地图数组, 1)
或 临时列 < 1 或 临时列 > 取数组下标 (地图数组, 2))

```

```

' 越界判断

```

```

    如果真 (地图数组 [临时行] [临时列] = 临时数据)

```

```

        ' 消块

```

```

        地图数组 [临时行] [临时列] = 0

```

```

        地图数组 [临时节点.行] [临时节点.列] = 0

```

```

        ' 以新找到的节点为基准, 再寻找它周围的相同块

```

```

        重定义数组 (消除节点组, 真, 取数组成员数 (消除节点组) + 1)

```

```

        消除节点组 [取数组成员数 (消除节点组)].行 = 临时行

```

```

        消除节点组 [取数组成员数 (消除节点组)].列 = 临时列

```

```

        ' 添加动画组对象

```

```

        重定义数组 (消除块动画组, 真, 取数组成员数 (消除块动画组) + 1)

```

```

        消除块动画组 [取数组成员数 (消除块动画组)].初始化数据 (图块动画组, 地图.取单元格横坐标 (临时列), 地图.取单元格纵坐标 (临时行), (临时数据 - 1) × #图块动画帧数 + 1, 临时数据 × #图块动画帧数, 块动画动作索引组, 1, 1)

```

```

        重定义数组 (消除块动画组, 真, 取数组成员数 (消除块动画组) + 1)

```

```

        消除块动画组 [取数组成员数 (消除块动画组)].初始化数据 (图块动画组, 地图.取单元格横坐标 (循环变量2), 地图.取单元格纵坐标 (循环变量1), (临时数据 - 1) × #图块动画帧数 + 1, 临时数据 × #图块动画帧数, 块动画动作索引组, 1, 1)

```

判断当前基准点的四个方向相邻格的值是否与它相等，相等意味着它们的图案相同，即满足消除条件。如果相邻的块和“当前块”相同，则将这个块的位置作为新的节点加入“消除节点组”以便继续探寻。同时在满足条件块的位置生成一个播放消除动画的对象。

```

' 让程序在下次循环时继续判断

```

```

移动中 = 真

```

```

本次消除块数 = 本次消除块数 + 1

```

```

    如果真 (选择框对象.取状态 () = 2 且 (选择框对象.取所在行 () = 临时行 且 选择框对象.取所在列 () = 临时列 或 选择框对象.取所在行 () = 循环变量1 且 选择框对象.取所在列 () = 循环变量2))

```

```

        ' 恢复选择框状态

```

```

        选择框对象.置状态 (1)

```

```

    如果真 (循环变量3 = 4)

```

```

        ' 遍历完4个方向后, 删除这个节点

```

```

        删除成员 (消除节点组, 当前节点索引, 1)

```

```

-- 计次循环尾 ()

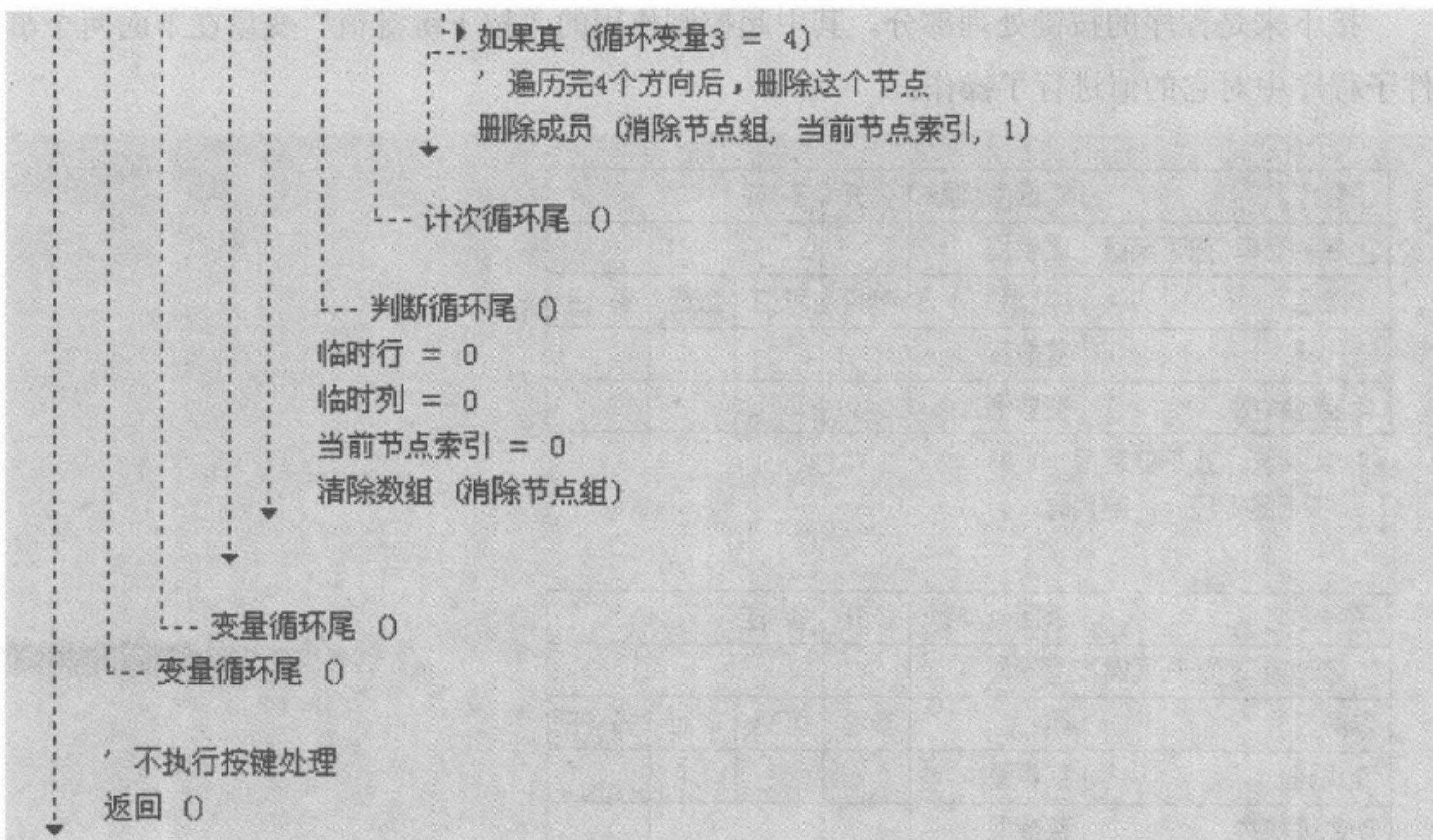
```

接下来，将块移动中的状态设为“真”，以便程序处理由于消除所引发的块下落或再



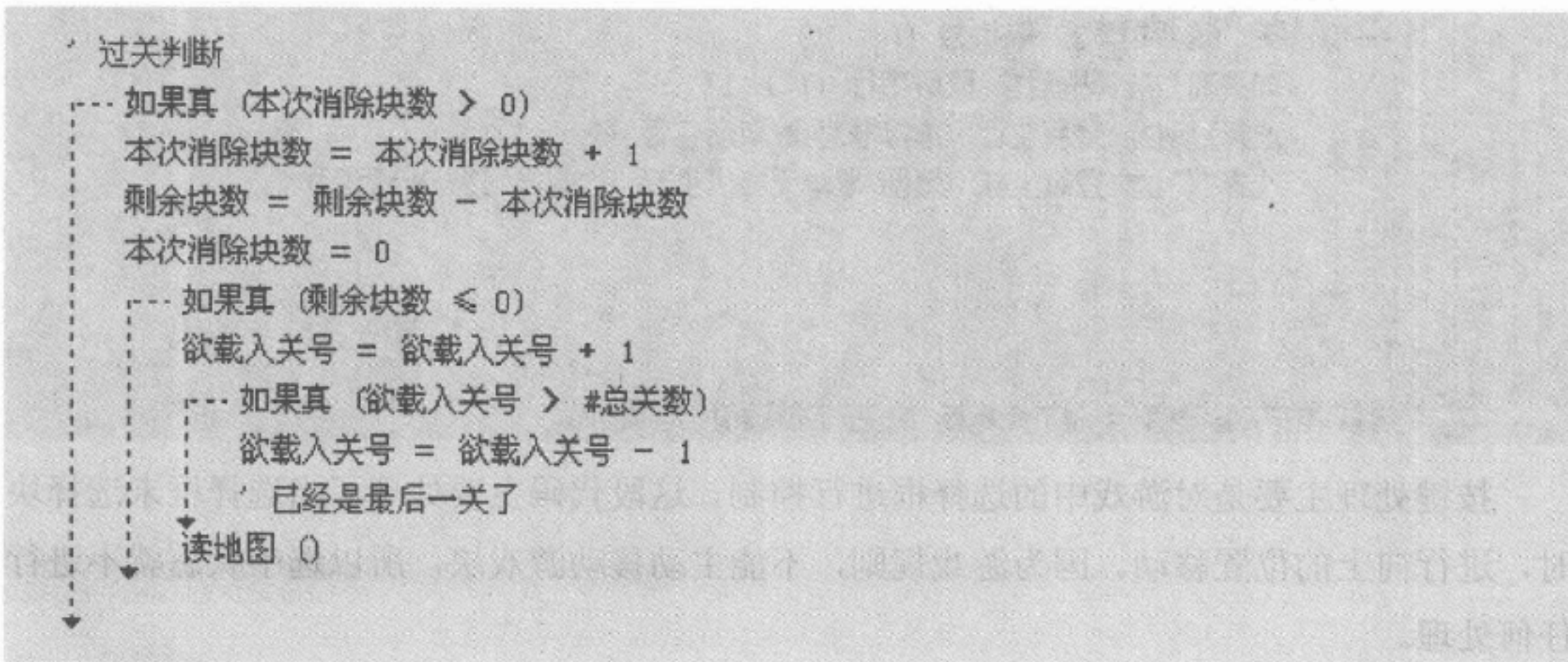


次消除的情况。如果选择框当前选择了被消除的块，则取消选择框的选择状态。最后判断“当前块”是否完成了对其四个方向的探寻，如果是则在需要探索的节点组中删除它。



当对某一块周围是否有相同块的处理结束后，将所用到的临时变量和对象恢复到初始值，以便下次循环对下一个块处理时使用。还记得判断下落和消除前我们判断了当前是否在下落的状态么？这里使用“返回”的意义在于，在处理下落或消除的本次游戏循环中，不执行按键处理的代码。因为本次游戏循环中只处理下落过程中的一格，并不是一次到位地将块移动到目标位置，需要在下一次游戏循环中继续处理，直到到目的位置。所以，如果在未完成移动或消除任务前去处理移动块，那么就有可能使游戏过程出现意外的情况。

在看过比较关键的下落和消除处理部分代码后，下面继续看游戏逻辑子程序中还做了哪些工作。







这段代码实现了过关后进入下一关的功能。判断是否过关的依据为，剩余的普通块数量是否为0。

接下来是程序的按键处理部分。其中起控制作用的“按下键键值”变量在下面两个事件子程序中对它的值进行了操作：

子程序名	返回值类型	公开	备注		
__启动窗口_按下某键	逻辑型				
参数名	类型	参考	可空	数组	备注
键代码	整数型				
功能键状态	整数型				

--- 如果真 (按下键键值 = 0)  
 按下键键值 = 键代码

子程序名	返回值类型	公开	备注		
__启动窗口_放开某键	逻辑型				
参数名	类型	参考	可空	数组	备注
键代码	整数型				
功能键状态	整数型				

--- 如果真 (键代码 = 按下键键值)  
 按下键键值 = 0

这样做的目的在于，每次游戏循环中只处理一个按键的功能。只有当按下的键抬起后，才能处理其他键的按下。

回到“游戏逻辑”子程序中：

#### 按键处理

```

--- 如果真 (按下键键值 ≠ 0)
  判断 (按下键键值 = #上光标键 或 按下键键值 = #键8)
  按下键键值 = 0
  如果真 (选择框对象.取状态 () = 1)
    如果真 (选择框对象.取所在行 () > 1)
      选择框对象.置所在行 (选择框对象.取所在行 () - 1)
      选择框对象.置纵坐标 (地图.取单元格纵坐标 (选择框对象.取所在行
        ()))
    判断 (按下键键值 = #下光标键 或 按下键键值 = #键2)
  
```

按键处理主要是对游戏中的选择框进行控制。这段代码主要处理了当选择框未选择块时，进行向上的位置移动。因为游戏规则，不能主动移动游戏块，所以选中状态就不进行任何处理。



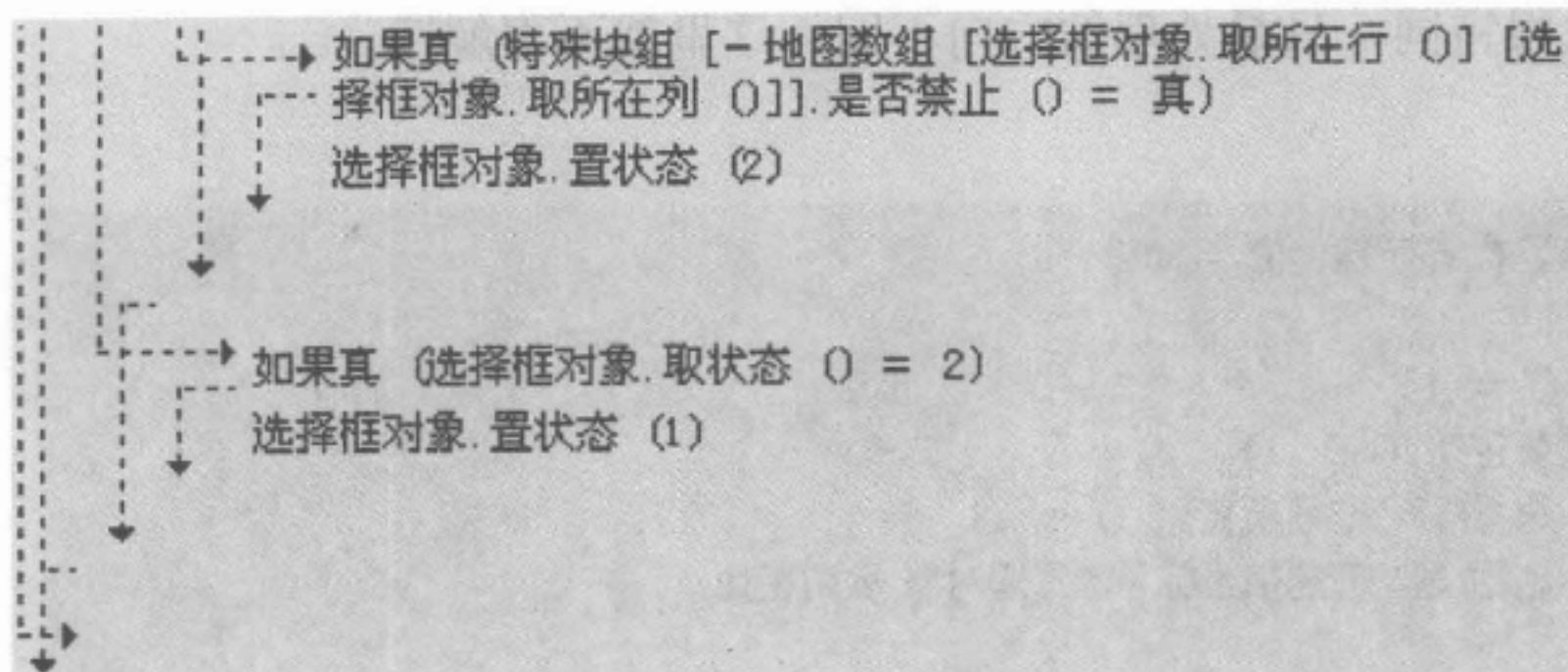


接下来看看向左移动:

最后是对选择框的选中状态的处理，代码如下：

更多书籍：[www.i-ebook.cn](http://www.i-ebook.cn)





如果当前选择框未选中，且选择格所在不是空格，那么继续处理。接下来判断当前所在格是不是一个普通格，如果是则改变选择框的状态为选中状态；如果不是，则再判断是否可以移动的特殊块，如果是，则选中。

取消选中很简单：如果当前选择框已选中，则取消选中。

到此整个“游戏逻辑”子程序就阅读完了，但在子程序一开头还留有一个问题——特殊块的处理。接下来就主要解读“特殊块类”和其几个关键的子类。

### 17.2.3 解读特殊块功能的实现

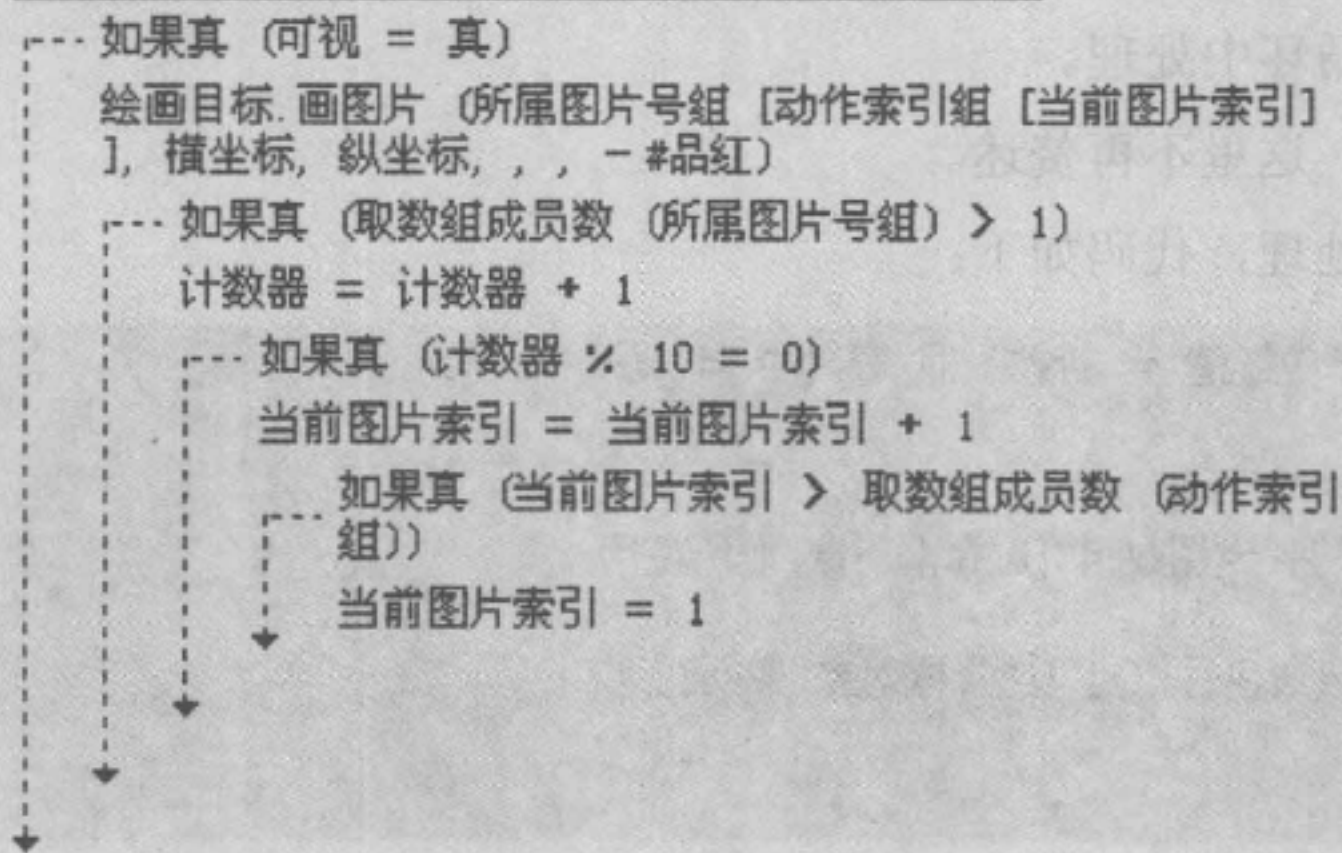
“特殊块类”作为各种拥有特殊性质和功能的图块的子类，提供了通用的输入输出接口和特殊块共有的属性和方法。比较关键的方法有“重画”、“置数据”、“更新状态”。

其中“重画”作为所有的特殊块共有方法，为特殊块提供了最基础的画面重画接口。

“置数据”和“更新状态”在“特殊块类”中为空方法，但所有子类必须重写这两个方法。其作用是提供统一的对外接口，但各子类中对该接口的实现可以不尽相同。

先来看“特殊块类”中的“重画”方法：

方法名	返回值类型	公开	备注		
重画		✓			
参数名	类型	参考	可空	数组	备注
绘画目标	画板				





如果当前特殊块对象是可视的，那么将当前对应的图片画到指定位置上，并将图片中的品红视为透明色。

接下来是对有多帧画面的特殊块（做对画出图片号组索引）的处理，由于刷新频率比较快，所以采用每重画十次再切换一次图片索引的方式。当有多帧动画时，通过动作索引组，可以指定按何种顺序播放图片号组中所对应的图片。

对“置数据”接口的实现，各特殊块的子类所做的操作都很相似。只是初始化数据时对应的类型不同，大家可以自行解读。

下面带着大家解读两个比较有特点的特殊块的子类，它们对“更新状态”接口的实现截然不同。

先来看“电梯\_垂直类”对“更新状态”接口的实现：

方法名	返回值类型	公开	备注
更新状态		✓	

变量名	类型	静态	数组	备注
临时行	整型			
临时数据	整型			
循环变量	整型			
上面块数据	整型			

```

计数器 = 计数器 + 1
-- 如果真 (计数器 % 10 ≠ 0)
-- 返回 0
                    
```

这段代码的作用是，每10个游戏循环做一次处理（避免执行太频繁，耗费大量资源）。

```

-- 如果 (移动速度 < 0)
-- 向上
-- 如果 (取所在行 0 > 1 且 地图数组 [取所在行 0 - 1] [取所在列 0] = 0)
-- 上边没块的情况
-- 累计移动长度 = 累计移动长度 + 移动速度
-- 如果真 (累计移动长度 ÷ #图块高度 < 0)
-- 移动电梯
-- 地图数组 [取所在行 0 - 1] [取所在列 0] = 地图数组 [取所在行 0] [取所在列 0]
-- 地图数组 [取所在行 0] [取所在列 0] = 0
-- 置所在行 (取所在行 0 - 1)
-- 累计移动长度 = 0
-- 上边有块的情况
                    
```

首先判断了当前电梯的移动方向，如果是向上运动，则进入判断块。如果电梯上面是空格，则调换两个格的成员数据，实现电梯上移一格的效果。





接下来看当前电梯上有格的情况：

```

┌─▶ 上边有块的情况
│   ' 判断顶层是否有空
│   临时行 = 取所在行 () - 1
│   --- 如果真 (临时行 < 1)
│       移动速度 = - 移动速度
│   └─▶ 判断循环首 (临时行 > 0)

```

首先从电梯所在的上一行开始判断，若上一行小于1，说明已经在整个地图的顶部，那么直接改变移动方向。

接下来的判断循环首体内的代码作用是：探寻电梯是否可以带着它上面的块向上移动。

```

┌─▶ 判断循环首 (临时行 > 0)
│   临时数据 = 地图数组 [临时行] [取所在列 ()]
│   --- 判断 (临时数据 > 0)
│       ' 普通块，进入下一次循环判断
│       临时行 = 临时行 - 1
│       到循环尾 ()
│   └─▶ 判断 (临时数据 = 0)

```

这段代码先判断当前电梯上方的格是否是普通块，如果是则继续判断它的上面一格。如果不是普通块，则会继续下面的处理：

```

┌─▶ 判断 (临时数据 = 0)
│   ' 有空
│   ' 带块上移
│   累计移动长度 = 累计移动长度 + 移动速度
│   --- 如果真 (累计移动长度 ÷ #图块高度 < 0)
│       变量循环首 (临时行, 取所在行 () - 1, 1, 循环变量)
│       地图数组 [循环变量] [取所在列 ()] = 地图数组 [循环变量 + 1] [取所在列 ()]
│       地图数组 [循环变量 + 1] [取所在列 ()] = 0
│       如果真 (选择框对象.取所在行 () = 循环变量 且
│           --- 选择框对象.取所在列 () = 取所在列 () 且 选择框对象.取状态 () = 2)
│               ' 移动选择框
│               选择框对象.置所在行 (选择框对象.取所在行 () - 1)
│               选择框对象.置纵坐标 (地图.取单元格纵坐标 (选择框对象.取所在行 ()))
│           --- 变量循环尾 ()
│       ' 移动电梯
│       置所在行 (取所在行 () - 1)
│       移动中 = 真
│       累计移动长度 = 0
│       跳出循环 ()
│   └─▶ 判断 (特殊块组 [- 临时数据].取块类型 () = -2)

```





这段代码首先判断临时位置是否为空格，如果是则说明可以带着电梯上的块进行上移。首先进行一些校验性的检查，然后通过改变地图数组成员值的方式，实现电梯上的块依次向上移的效果。如果选择框当前选中了被移动的块，则更新选择框的位置。最后改变电梯自身的位置并开启下落及移动处理。

如果临时位置不是空格，则会继续下面的处理：

```

--> 判断 (特殊块组 [- 临时数据].取块类型 () = -2)
    ' 是石头
    --> 如果 (临时行 > 1)
        上面块数据 = 地图数组 [临时行 - 1] [取所在列 ()]
        ' 判断石头上面是否有块
        --> 如果 (上面块数据 ≠ 0)
            ' 有块则不动, 改变移动方向
            移动速度 = - 移动速度
        --> ' 没有则块带石头上移

```

这段代码处理了当电梯上有石头的情况。如果石头上面还可能有块，则获取石头上面块的信息。如果石头上有块，则不移动电梯，直接改变电梯的移动方向。

```

--> ' 没有则块带石头上移
    累计移动长度 = 累计移动长度 + 移动速度
    --> 如果真 (累计移动长度 ÷ #图块高度 < 0)
        --> 变量循环首 (临时行, 取所在行 (), 1, 循环变量)
            ' 调换数据
            上面块数据 = 地图数组 [循环变量] [取所在列 ()]
            地图数组 [循环变量 - 1] [取所在列 ()] = 上面块数据
            地图数组 [循环变量] [取所在列 ()] = 0

            --> 如果真 (上面块数据 < 0)
                ' 更新特殊块位置
                特殊块组 [- 上面块数据].置所在行 (循环变量 - 1)

            --> 如果真 (选择框对象.取所在行 () = 循环变量 且 选择框对象.取所在列 () = 取所在列 () 且 选择框对象.取状态 () = 2)
                ' 移动选择框
                选择框对象.置所在行 (选择框对象.取所在行 () - 1)
                选择框对象.置纵坐标 (地图.取单元格纵坐标 (选择框对象.取所在行 ()))

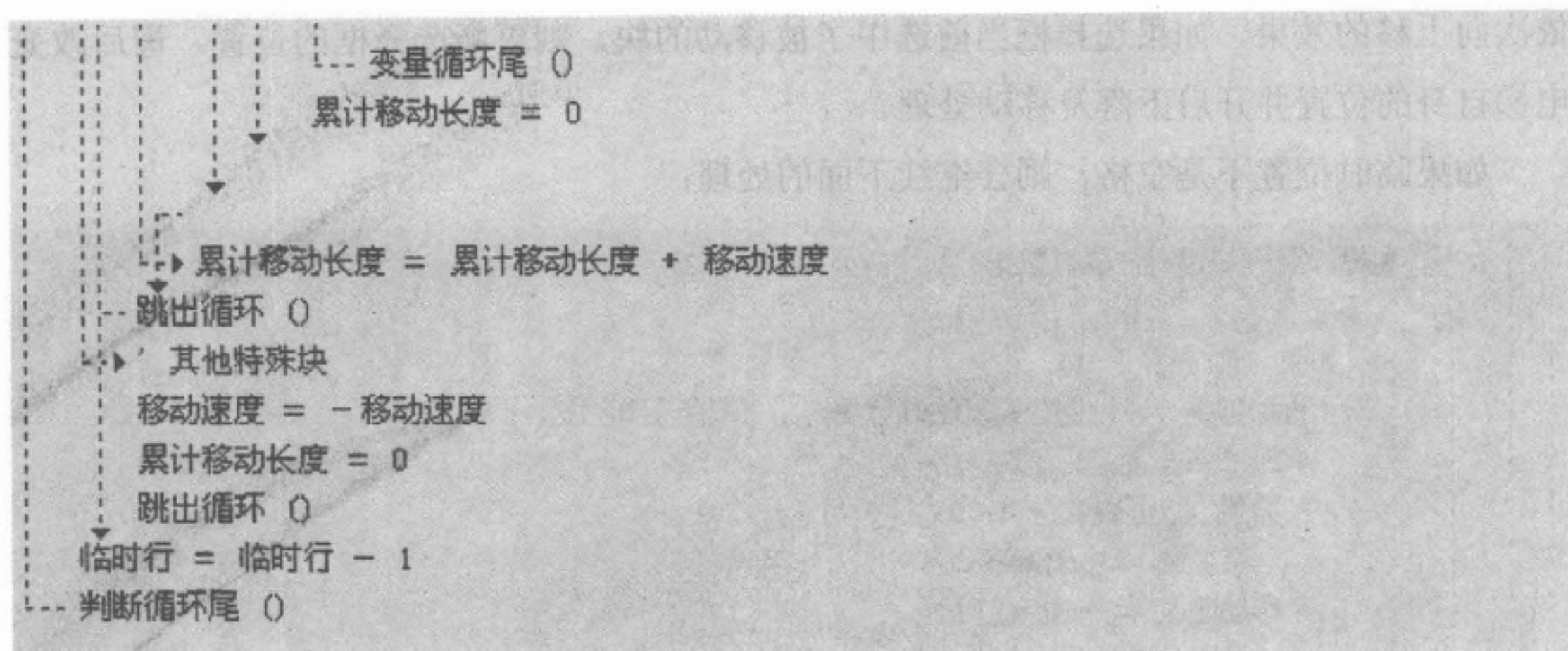
            --> 变量循环尾 ()
        累计移动长度 = 0

```

如果石头上面是空格，则电梯带着石头上移。实现思想跟移动普通块相同。

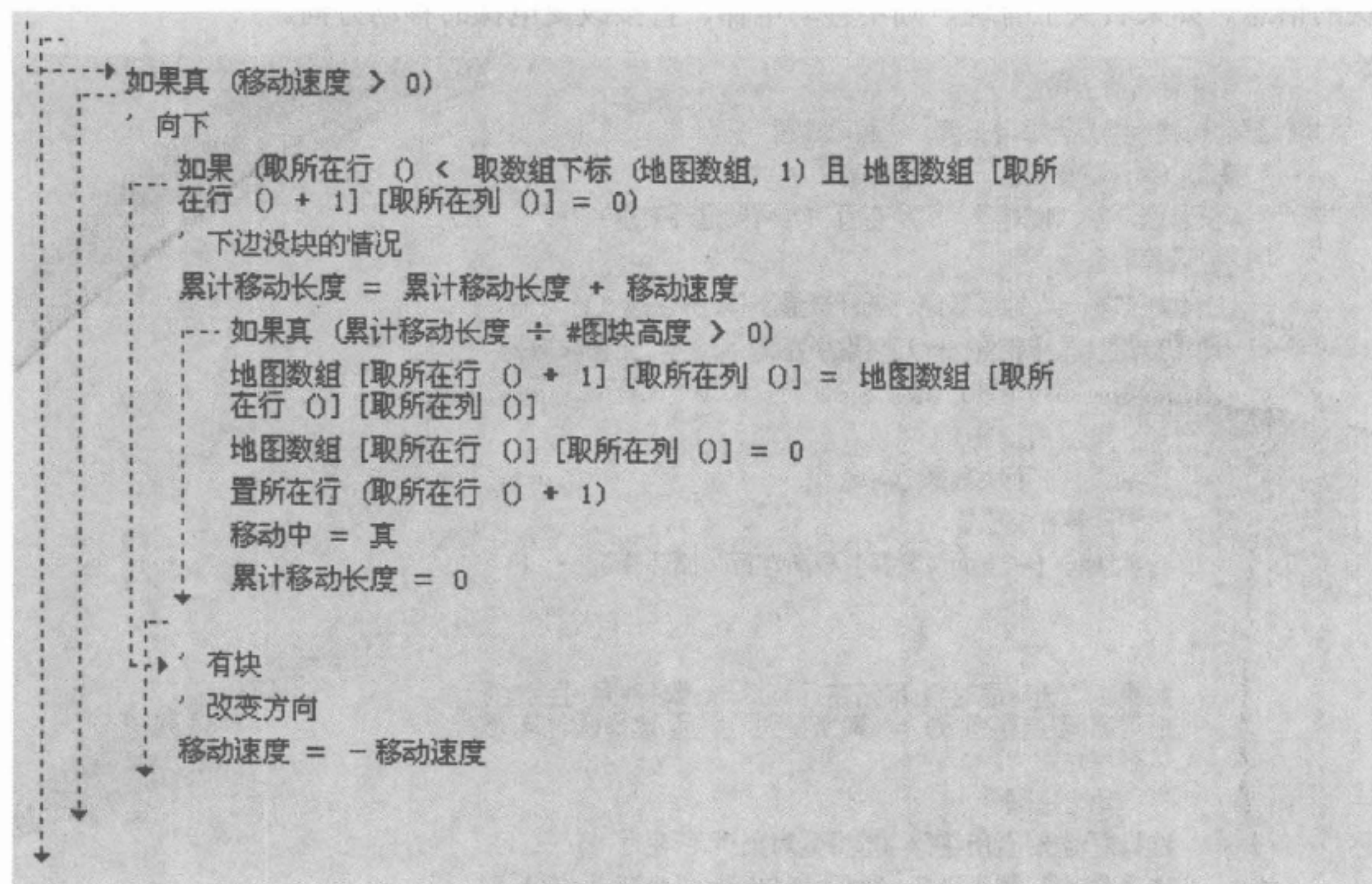


如果电梯上方不是空，不是普通块，也没到地图顶端即说明是其他类型的特殊块，进入下面的处理：



如果电梯不能向上移动，则改变方向。

接下来看电梯向下时的操作：



首先，判断电梯下方是否是空格，如果是则通过改变数组成员值实现电梯的下移效果。更新当前电梯对象的位置并开启下落及消除处理。如果电梯下方有块，则直接改变电梯的移动方向。

到这，“电梯\_垂直类”对“更新状态”方法的实现全部解读完了。接下来再看看“火焰类”对“更新状态”方法的实现。



方法名	返回值类型	公开	备注
更新状态		✓	

变量名	类型	静态	数组	备注
临时数据	整型			

```

--- 如果真 (取所在行 0 > 1)
    临时数据 = 地图数组 [取所在行 0 - 1] [取所在列 0]
    --- 如果真 (临时数据 ≠ 0)
        --- 如果 (临时数据 > 0)
            ' 普通块, 游戏结束
            游戏结束 = 真
            地图数组 [取所在行 0 - 1] [取所在列 0] = 0
            --- 如果真 (特殊块组 [-临时数据]. 取块类型 0 = -2)
                ' 石头
                地图数组 [取所在行 0 - 1] [取所在列 0] = 0
                特殊块组 [-临时数据]. 置可视 (假)
            暂停 0
        --- 如果真 (选择框对象. 取所在行 0 = 取所在行 0 - 1 且 选择框对象. 取所在列 0 = 取所在列 0 且 选择框对象. 取状态 0 = 2)
            选择框对象. 置状态 (1)
    暂停 0

```

“火焰类”中的“更新状态”主要实现了对普通块和特殊块在火焰块上的不同处理。首先判断火焰块上是否是普通块，如果是则闯关失败；如果是石头则通过改变石头的可视状态实现消除石头的效果。由于游戏在这个版本中，只有普通块和石头可以下落和移动，所以只对这两种块做了相应的处理。接下来是处理选择框当前选中被“烧掉”的块，则改变它的选中状态。

通过对两个特殊块子类同一接口的实现进行解读发现，虽然接口相同，但所做的操作却相差甚远。为什么要这样设计程序呢？这样做的好处在哪里？

这样设计是使用了面向对象的编程思想。这样做的好处是，每一类物体都按照自己的算法去处理自己的事务，且通过统一的接口可以方便地管理它们。

还记得“游戏逻辑”子程序开头吗？

```

' 更新特殊块数据
--> 变量循环首 (取数组成员数 (特殊块组), 1, -1, 循环变量1)
    特殊块组 [循环变量1]. 更新状态 0
-- 变量循环尾 0

```





通过一个循环，调用各特殊块的统一接口，就可以实现各种特殊块的状态更新任务及特殊块在每个游戏循环所做的处理。

### 17.2.4 解读重绘画面

再好的游戏逻辑，再优秀的程序设计如果不能将最终效果展现给玩家，那一切都是没有意义的。所以游戏画面的处理也是游戏中相当重要的一环。下面就从“重绘画面”子程序起，解读“对对碰”中画面的处理。

子程序名	返回值类型	公开	备注
重绘画面			

变量名	类型	静态	数组	备注
循环变量	整数型			

```
' 清屏
后台缓冲.刷子颜色 = #白色
后台缓冲.填充矩形 (0, 0, 后台缓冲.取用户区宽度 (), 后台缓冲.取用/
高度 ())
' 画背景
' 画地图
地图.重画 (后台缓冲)
' 画特殊块
--> 计次循环首 (取数组成员数 (特殊块组), 循环变量)
    特殊块组 [循环变量].重画 (后台缓冲)
-- 计次循环尾 ()
' 画消除块动画
--> 变量循环首 (取数组成员数 (消除块动画组), 1, -1, 循环变量)
    -- 如果 (消除块动画组 [循环变量].是否已停止 () = 真)
    -- 删除成员 (消除块动画组, 循环变量, 1)
    --> 消除块动画组 [循环变量].重画 (后台缓冲)
-- 变量循环尾 ()
' 画选择框
选择框对象.重画 (后台缓冲)
```

“重绘画面”子程序中，主要做了两件事：清除前一次的旧画面和按照最新的状态重绘画面。

其中“清屏”很简单，就是使用白色填充整个后台缓冲；在前一节中对特殊块的重画处理也已经解读过了，所以接下来主要带大家解读地图、动画块和选择框的重画方法。

我们先看地图的重画方法（对应地图类的重画方法）：



方法名	返回值类型	公开	备注		
重画		✓			
参数名	类型	参考	可空	数组	备注
绘画目标	画板				

变量名	类型	静态	数组	备注
所需画出横向起始索引	整数型			
所需画出横向结束索引	整数型			
所需画出纵向起始索引	整数型			
所需画出纵向结束索引	整数型			
画出横向位置	整数型			
画出纵向位置	整数型			
循环变量1	整数型			
循环变量2	整数型			

所需画出横向起始索引 = (0 - 地图横坐标) ÷ 小格宽  
 所需画出横向结束索引 = 所需画出横向起始索引 + 绘画目标.取  
 用户区宽度 () ÷ 小格宽  
 所需画出纵向起始索引 = (0 - 地图纵坐标) ÷ 小格高  
 所需画出纵向结束索引 = 所需画出纵向起始索引 + 绘画目标.取  
 用户区高度 () ÷ 小格高

```

-- 如果真 (所需画出横向起始索引 < 1)
    所需画出横向起始索引 = 1
-- 如果真 (所需画出纵向起始索引 < 1)
    所需画出纵向起始索引 = 1
-- 如果真 (所需画出横向结束索引 > 横向小格数)
    所需画出横向结束索引 = 横向小格数
-- 如果真 (所需画出纵向结束索引 > 纵向小格数)
    所需画出纵向结束索引 = 纵向小格数
    
```

由于画面的重绘相对来说是比较耗费资源的，所以方法内一开始先计算需要重绘的部分，不需要重绘或不显示的部分就可以跳过重绘过程来减少资源的消耗。

```

画地图
-- 如果 (地图纵坐标 > 0)
    画出纵向位置 = 地图纵坐标
    画出纵向位置 = 地图纵坐标 % 小格高

    变量循环首 (所需画出纵向起始索引, 所需画出纵向结束索引,
    1, 循环变量1)

        画出横向位置 = 地图横坐标

        -- 如果 (地图横坐标 > 0)
            画出横向位置 = 地图横坐标
            画出横向位置 = 地图横坐标 % 小格宽
    
```



```

--> 变量循环首 (所需画出横向起始索引, 所需画出纵向结束索引, 1, 循环变量2)
-- 如果真 (地图数组 [循环变量1] [循环变量2] > 0)
    绘画目标. 画图片 (图块图片号组 [地图数组 [循环变量1] [循环变量2]], 画出横向位置, 画出纵向位置, , , #拷贝)
    画出横向位置 = 画出横向位置 + 小格宽
-- 变量循环尾 0
画出纵向位置 = 画出纵向位置 + 小格高
-- 变量循环尾 0

```

这段代码主要是通过两层循环遍历“地图数组”成员值，并将成员值所对应的图块画到相应位置处。需要说明的是，对地图横/纵坐标不大于0时的处理思想。如果地图的横/纵坐标不大于0，说明地图的横/纵坐标有可能小于或等于0。如果是等于0的情况，那么通过取余，仍会得到0，即画出位置会和横/纵坐标0点重合；小于0的情况说明地图有一部分是看不见的，显示的可能是小格的一半（横/纵，对应左/上半），那么起始画出位置就应当是一个绝对值小于小格宽的负坐标。通过对当前横/纵坐标取小格的宽/高，正好实现了这一需求。

接下来再来看看消块动画的重画处理（对应精灵类的重画方法）：

方法名	返回值类型	公开	备注		
重画		✓			
参数名	类型	参考	可空	数组	备注
绘画目标	画板				

绘画目标. 画图片 (图片号组 [动作索引组 [当前图片索引]], 横坐标, 纵坐标, , , -#品红)

```

-- 如果真 (停止 = 假)
    计数器 = 计数器 + 1
    -- 如果真 (计数器 % 帧率 = 0)
        当前图片索引 = 当前图片索引 + 1
        -- 如果真 (当前图片索引 > 取数组成员数 (图片号组))
            当前图片索引 = 1
            -- 如果真 (循环次数 > 0)
                循环次数 = 循环次数 - 1
            -- 如果真 (循环次数 = 0)
                停止 = 真

```

没有什么难懂的算法，其结构和“特殊块类”的重画方法相同，这里不再赘述。



再来看看选择框的重画方法：

方法名	返回值类型	公开	备注		
重画		✓			
参数名	类型	参考	可空	数组	备注
绘画目标	画板				

根据当前状态重画选择框

绘画目标.画图片 (图片号\_选择框 [状态], 实际画出横坐标, 实际画出纵坐标, , , - #蓝色)

选择框类的重画方法很简单，只有一句代码——按照选择框的状态画出相应的图片。但这里需要解释一下画出的坐标。

选择框画出时的“实际画出横/纵坐标”并不是其所在小格左上角的坐标。其根本原因在于，每个小格的图片宽高是20×20，但选择框图片却是38×26（其框内尺寸为20×20，正好框住小块）。所以，画出时的“实际画出横/纵坐标”是经过处理的。处理它们的代码如下：

方法名	返回值类型	公开	备注
计算偏移值			

实际画出横坐标 = 横坐标 - 横向偏移量

实际画出纵坐标 = 纵坐标 - 纵向偏移量

将当前所在小格的坐标通过减去对应的偏移量，得到实际应画出的坐标。偏移量是怎样得到的呢？其实就是选择框的图片（宽/高）减去小格的图片（宽/高）再除以2（为了使小格在选择框的中间）。

到这里，对游戏“对对碰”关键部分代码的解读就全部完成了，下面来总结一下它的优点和缺点。

“对对碰”游戏的优点在于，使用了对事物功能抽象化的面向对象程序设计思想，这样做的好处在于对某一类事物可以方便地进行管理。另外游戏的框架及逻辑处理、画面处理的实现也很值得学习。

“对对碰”游戏的缺点有：程序算法还不够高效，还可以继续优化。比如在探寻相同块时，对四个方向全部探寻就显得没有必要（因为由一个块再探寻，肯定至少有一个方向是探寻过相同块）。垂直电梯向下移动时，没有必要每次都进行下落和消除处理。

最后，再介绍一些阅读程序的经验：

（1）跟着程序的运行流程阅读。

（2）若发现所用元素（变量、子程序调用等）意义不明，可以使用“全局查找”功能找到它的定义、初始化、主要实现、等位置；也可以使用断点调试的方法来获取元素当前的状态，最终确定该元素的意义。



(3) 当发现有意义不明确, 且不会影响所要阅读代码的主要功能时, 可先跳过该段代码。

(4) 当发现意义不明确, 且会影响所要阅读代码的主要功能时, 应全面阅读相关代码后, 对该段代码的意义进行判断。

## 17.3 后记

学习软件编程是一个循序渐进积累经验的过程, 不存在绝对的捷径, 但通过选择正确的教材和使用正确的学习方法可以更快地跨越编程的门槛。

那怎样来学习才算正确的方法呢?

- (1) 学习一门功能强大且简单实用的编程语言。
- (2) 多看别人的代码, 学习解决问题的思路(程序的算法)。
- (3) 广泛地了解各类相关(软/硬)知识。
- (4) 多动手自己编写软件, 重点攻克感兴趣的领域。

编写软件时要注意的几点(也算是好习惯):

- (1) 变量、子程序、组件的命名要简洁且有意义。
- (2) 多写程序注释。
- (3) 尽量进行开发文档和软件需求的编写。
- (4) 碰到难点时不要钻牛角尖, 多换几种思路, 可能会豁然开朗。
- (5) 对程序的代码、编译的目的程序进行完善的版本号或修改日期的管理, 并多做资料的备份。
- (6) 多与他人交流(如去论坛或和朋友交流)经验。

附: 易语言相关网络资源:

易语言网站: <http://www.dywt.com.cn/>; <http://www.eyuyan.com/>

易语言论坛: <http://bbs.eyuyan.com/>

易语言资源网: <http://www.wodesoft.com/>

易语言.飞扬: <http://dotef.eyuyan.com/>

易乐谷: <http://www.dywt.com.cn/elogol/>

EPL (Easy Programming Language): <http://epl.eyuyan.com/>